Retro Toolbox Manual

Chapter 2

Operator Descriptions

Chapter 2 - Operator Descriptions

A. Introduction

This chapter contains the man1 pages for all programs in the Retro toolbox, organized alphabetically by object name. Any man1 page can be accessed by typing the following command.

```
% kman <program_name>
```

This information can also be accessed by selecting the *Help* button on the program's graphical user interface, which is displayed when running the program from within cantata, or when running it from the command line using the *-gui* option.

The keywords (-k) option in kman allows the user to provide a keyword string for searching. When invoked with the *-k* option, kman will perform the search and provide a list of the names and short descriptions of programs found by the search.

```
% kman -k "keyword string"
```

If you wish to display only the command line usage information for a program, enter the program name, followed by -U.

```
% <program_name> -U
```

B. Khoros 1.0 Image Processing & Analysis Operators

B.1. lrfclass — *Classify Image Using the Localized Receptive Field Classifier (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Classification*
Operator:	*LRF-classifier*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

lrfclass classifies an image using the Localized Receptive Field classifier (LRF). The Localized Receptive Field (LRF) is based on a single layer of self-organizing, "localized receptive field" units, followed by a single layer perceptron. The single layer of perceptron units use the LMS or Adaline learning rule to adjust the weights. The weights are adjusted or "trained" using the companion program, "lrftrain". After training the weights, using the "lrftrain" program, a number of similar images may be quickly classified with this program based on the training data set.

LRF network theory

The basic network model of the LRF consists of a two layer topology. The first layer of "receptive field" nodes are trained using a clustering algorithm, such as K-means, or some other algorithm which can determine the receptive field centers. Each node in the first layer computes a receptive field response function, which should approach zero as the distance from the center of the receptive field is increased. The second layer of the LRF model sums the weighted outputs of the first layer, which produces the output or response of the network. A supervised LMS rule is used to train the weights of the second layer nodes.

The response function of the LRF network is formulated as follows:

f(x) = SUM(Ti * Ri(x))

where,

Ri(x) = Q(||x - xi|| / Wi)

x - is a real valued vector in the input space,
Ri - is the ith receptive field response function,
Q - is a radially symmetric function with a single maximum at the origin, decreasing to zero at large radii,
xi - is the center of the ith receptive field,
Wi - is the width of the ith receptive field,
Ti - is the weight associated with each receptive field.

The receptive field response functions (Ri(x)), should be formulated such that they decrease rapidly with increasing radii. This ensures that the response functions provide highly localized representations of the input space. The response function used in this algorithm is modeled after the Gaussian, and uses the trace of the covariance matrix to set the widths of the receptive field centers.

Prior to using this algorithm, it is necessary to "train" the weights for the output layer by running the companion program, "lrftrain", on a previously clustered image. Thus the inputs to this program are the original input image (-i1), the cluster center image (-i2), the cluster variance image (-i3), and the weight image (-i4). The original input image contains all of the features used in the original clustering (ie. vkmeans). The cluster center image (-i2) contains the locations of the cluster centers in the input feature space, which fixes the centers of the localized receptive fields. The cluster variance image (-i3) contains the variances associated with each cluster center. This establishes the widths of the localized receptive field Gaussians. The weight image (-i4) contains all of the weights for each node in the output layer.

The number of receptive field response nodes in the first layer of the LRF is determined by the number of cluster centers in the "cluster center" image. The number of output classes, and hence the number of output nodes in the second (ie. last) layer, is determined by the number of desired classes that was specified in the "supervised" classification phase of the clustering. This information is contained in the last band of the cluster center image. The number of weights in the network is determined by the number of receptive field response nodes and the number of output nodes. That is,

#Wts = (#rf_response_nodes * #output_nodes) + #output_nodes

The resulting output image is classified with the desired number of classes specified in the last band of the "cluster center" (-i2) image. The number of desired classes corresponds to the number of output nodes in the last layer of the LRF network. This classified image is of data storage type INTEGER.

Input Options

-b is an integer that specifies the border width, in pixels, encompassing the desired region of the image to be classified. This region is ignored during the classification process.

This routine was written with the help of and ideas from Dr. Don Hush, University of New Mexico, Dept. of EECE.

Examples

lrfclass -i1 feature_image.xv -i2 cluster_centers -i3 variances -i4 weight_image -o classified_image -b 4

This example uses feature_image.xv as the input feature image, and the three other images from the companion program "lrftrain". These include the cluster_centers image, the variances image, and the weight_image. The resulting classified image is stored as "classified_image". A border width of 4 pixels is specified, which will cause the outermost four pixels of the image to be ignored.

See Also

lrftrain(1)

Restrictions

All input images MUST be of data storage type FLOAT. The resulting classified image (-o) is of data storage type INTEGER.

Command Line Arguments

```
Usage for lrfclass: Classify Image Using the Localized Receptive Field Classifier  (K1)
% lrfclass
  -i1        (infile)  input image
  -i2        (infile)  cluster center image
  -i3        (infile)  cluster variance image
  -i4        (infile)  weight image
  -o         (outfile) classified image

  [-b]       (integer) Border Width
             (0 <= b <= 100, default = 0)
```

B.2. lrftrain — *Calculate Weights for Localized Receptive Field Classifier (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Classification*
Operator:	*LRF-training*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

lrftrain trains on an image for the weights used with the Localized Receptive Field classifier (see lrf-class). The Localized Receptive Field (LRF) is based on a single layer of self-organizing, "localized receptive field" units, followed by a single layer perceptron. The single layer of perceptron units use the LMS or Adaline learning rule to adjust the weights.

In contrast, multi-layer network models, such as the multi-layer perceptron (MLP), use the back propagation learning model. The back propagation learning model is based on a gradient descent procedure, which tends to converge at a very slow rate. Back propagation is also hampered by the fact that all layers of weights in the network are computed by minimizing the error, which is a function of the output. This tends to slow the learning, since all weights in the network must be determined with each iteration.

The LRF tries to overcome some of these problems by using a localized representation of the input space to limit the number of units that respond to a given input. This allows the LRF to converge at a faster rate than similar types of neural network models, since only those receptive fields which respond to an input need to be updated. Another factor that allows the LRF to reduce the learning time, is that it makes use of self-organized learning techniques, such as K-means, to train the receptive field centers.

LRF network theory

The basic network model of the LRF consists of a two layer topology. The first layer of "receptive field" nodes are trained using a clustering algorithm, such as K-means, or some other algorithm which can determine the receptive field centers. Each node in the first layer computes a receptive field response function, which should approach zero as the distance from the center of the receptive field is increased. The second layer of the LRF model sums the weighted outputs of the first layer, which produces the output or response of the network. A supervised LMS rule is used to train the weights of the second layer nodes.

The response function of the LRF network is formulated as follows:

f(x) = SUM(Ti * Ri(x))

where,

Ri(x) = Q(‖x - xi‖ / Wi)

x - is a real valued vector in the input space,
Ri - is the ith receptive field response function,
Q - is a radially symmetric function with a single maximum at the origin, decreasing to zero at large radii,
xi - is the center of the ith receptive field,
Wi - is the width of the ith receptive field,
Ti - is the weight associated with each receptive field.

The receptive field response functions (Ri(x)), should be formulated such that they decrease rapidly with increasing radii. This ensures that the response functions provide highly localized representations of the input space. The response function used here is modeled after the Gaussian, and uses the trace of the covariance matrix to set the widths of the receptive field centers.

The weights for the output layer are found using the LMS learning rule. The weights are adjusted at each iteration to minimize the total error, which is based on the difference between the network output and the desired result.

The key element to the success of the LRF is the self-organizing receptive fields. As noted above, the receptive field centers can be determined from a statistical clustering algorithm such as K-means. The inputs to the training phase of the LRF (ie. "lrftrain"), are the outputs from "vkmeans" and the original image. Specifically, these are the original input image, which may be a multi-band image containing all of the feature bands used in the classification, and the "cluster number" image, the "cluster center" image, and the "cluster variance" image. The "cluster number" image specifies which vector belongs to what cluster, the "cluster center" image specifies the cluster center locations in the feature space, and the "cluster variance" image specifies the variances of the data associated with each cluster center.

Prior to using the LRF algorithm, it is necessary to run "vkmeans" on the input training image to fix the cluster centers, followed by a supervised classification of the clustered image, which assigns a desired class to each cluster center. NOTE that the image resulting from the supervised classification MUST be appended to the "cluster center" image before running the LRF. This is necessary since it makes the appropriate desired class assignments to the cluster centers for the training phase of the LRF.

Input Options

-d is an integer specifying the iteration interval used to print the mean squared error (MSE) to the output statistics file. If this value is left at zero (the default), only the MSE of the first iteration is written to the file. Any other integer will cause the value of the MSE to be written to the statistics file at the iteration interval specified.

-cv is a float value that specifies the convergence value for the algorithm. When the current MSE value reaches the specified convergence value, the algorithm will terminate.

-meu is a float value that specifies the weight update parameter for the learning algorithm. This value can be adjusted from 0 to 1. NOTE: this parameter may have a significant affect on the rate of learning, and it may have to be adjusted several times to get a feel for the optimum learning rate.

-n is an integer that specifies the maximum number of iterations that the algorithm will run before terminating. It is initially set to an arbitrarily large number to allow the algorithm to complete the learning phase.

-delta is a float value that specifies the minimum change in the MSE value from one iteration to the next. This parameter may be used to terminate the algorithm when the change in the MSE is zero or very small, but the MSE has not yet reached the specified convergence value (-cv). This may occur when the learning has reached a "plateau" or "bench" and is no longer learning.

-b is an integer that specifies the border width, in pixels, encompassing the desired region of the image to be classified. This region is ignored during the classification process.

Of the four input images to this routine, all but the "cluster number" image must be of data storage type FLOAT. The "cluster number" image should be of data storage type INTEGER. The output "weight" image is written out as data storage type FLOAT. The output statistics file is stored as an ASCII file.

The statistics output file (-f) contains the following information:

MSE at the first iteration
MSE at each specified interval (optional)
Total Number of Iterations
Final MSE at termination of the algorithm
Convergence Parameter used (-cv)
Weight Update Parameter used (-meu)
Minimum Delta MSE value (-delta)
Border Width (-b)
Number of Response Nodes in the network
Number of Output Classes in the network

The number of receptive field response nodes in the first layer of the LRF is determined by the number of cluster centers in the "cluster center" image. The number of output classes, and hence the number of output nodes in the second (ie. last) layer, is determined by the number of desired classes that was specified in the "supervised" classification phase of the clustering. This information is contained in the last band of the cluster center image. The number of weights in the network is determined by the number of receptive field response nodes and the number of output nodes. That is,

#Wts = (#rf_response_nodes * #output_nodes) + #output_nodes

Advice

As an initial step, try running the algorithm with a small number of iterations (ex. -n = 500) to get a feel for how the MSE is behaving (ie. decreasing rapidly, slowly, or increasing). Make sure you have the MSE display parameter set to a reasonable interval (ex. -d = 10) so that you can see how the MSE is behaving. These values will be written to the statistics file (-f).

After you get an idea of how the MSE is behaving, set the convergence value (-cv) to a reasonable value. You may also try decreasing the weight update parameter (-meu) to learn at a slower rate. Often times a large weight update parameter will cause the learning to "oscillate" and never reach a small MSE. You may also want to set the minimum delta MSE parameter (-delta) to a small value, to ensure that the algorithm terminates if the MSE levels off.

This routine was written with the help of and ideas from Dr. Don Hush, University of New Mexico, Dept. of EECE.

Examples

lrftrain -i1 feature_image.xv -i2 cluster_centers -i3 variances -i4 cluster_numbers -o weight_image -f stats -d 10 -n 500

This example illustrates a good initial step at training on an image. The display interval is set to write out the MSE every 10 iterations. The number of iterations is set to a small value, 500, to ensure that the algorithm will stop in a reasonable amount of time to get a feel for how the MSE is behaving.

See Also

lrfclass(1)

Restrictions

All input images except the "cluster number" image (-i4) MUST be of data storage type FLOAT. The "cluster number" image (-i4) MUST be of data storage type INTEGER. The output "weight" image (-o) is of data storage type FLOAT.

Command Line Arguments

```
Usage for lrftrain: Calculate Weights for Localized Receptive Field Classifier  (K1)
% lrftrain
  -i1        (infile)  input image
  -i2        (infile)  cluster center image
  -i3        (infile)  cluster variance image
  -i4        (infile)  cluster number image
  -o         (outfile) weight image
  -f         (outfile) output file for training statistics

  [-b]       (integer) Border Width
             (0 <= b <= 100, default = 0)
  [-cv]      (float)   convergence value
             (cv > 0.0, default = 0.1)
```

```
  [-meu]     (float)   weight update value
             (0 <= meu <= 1, default = 0.5)
  [-n]       (integer) Maximum number of iterations
             (n > 0, default = 10000000)
  [-delta]   (float)   Min delta MSE value
             (delta > 0.0, default = 1e-09)
  [-d]       (integer) MSE display interval
             (d >= 0, default = 0)
```

B.3. v1bgamut — *Compress Grey Levels of Image (Produces Image with Map) (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Segmentation*
Operator:	*Compress Grey*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

NOTE: Program will NOT execute! v1bgamut functionality is replaced by the IMAGE::igamut operator. No command line argument changes are required to get the same result.

v1bgamut takes a 1 band generates a 1 band image with a map of the specified number of color levels that can be displayed using editimage. The object is to make the displayable image look like the original even though the number of color levels is greatly restricted.

The method employed is that used for igamut(1), but only for one band. All discussion pertinent to igamut(1) is applicable here, except that the precision is fixed at 8 bits (i.e. full precision).

Examples

v1bgamut -i image:moon -o compressed.xv -n 72

Compresses the grey levels of image "image:moon" and produces an image with a map containing 72 levels that approximates the original.

See Also

IMAGE::igamut(1)

Restrictions

v1bgamut works only on BYTE image with 1 band of data and no map.

Command Line Arguments

```
Usage for v1bgamut: Compress Grey Levels of Image (Produces Image with Map)  (K1)
% v1bgamut
  -i         (infile)  input image filename
  -o         (outfile) output image filename

  [-n]       (integer) specifies the number of levels to compress to
```

```
             (1 <= n <= 256, default = 128)
  [-r]       (float)   fraction of level splits based on 2-norm
             (0 <= r <= 1, default = 0.5)
```

B.4. vbandsp1 — *Extract a Specified Data Band from a Multiband Image (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Organize Data*
Operator:	*Extract 1 Band*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

NOTE: Program will NOT execute! vbandsp1 functionality is replaced by the DATAMANIP::kextact operator. To get the same result using kextract, set the elements size to 1 (-esize 1), and the elements offset (-eoff) to the old band number (-b).

vbandsp1 extracts a specified band from an image and creates a new image containing only that band. The map is also carried along with the new image, the map scheme will become ONEPERBAND regardless of the input image's map scheme. The comment field of the new image will read the same as the comment field of the input image, except it is append with the band number and the filename from which the band was taken.

The data band number option determines which data band to extract. NOTE: band numbering begins with 0.

Examples

vbandsp1 -i ball.rgb.xv -o ball.b.xv -b 2

This will extract the third band (the band representing blue), from the 3 band rgb input image. The result is a 1 band image stored as ball.b.xv.

See Also

DATAMANIP::kextract, RETRO::vbandsp3

Command Line Arguments

```
Usage for vbandsp1: Extract a Specified Data Band from a Multiband Image  (K1)
% vbandsp1
  -i         (infile)  input image filename
  -o         (outfile) output image filename

  [-b]       (integer) Data band number to extract
             (b >= 0, default = 0)
```

B.5. vbandsp3 — *Extract 3 Bands from a Multiband Image (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Organize Data*
Operator:	*Extract 3 Bands*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vbandsp3 extracts 3 bands from an image and creates new images containing only one band. The 3 bands extracted start with the band number Specified by the starting band number option. The band numbering scheme starts at zero.

The map is carried along with the new image, the map scheme will become ONEPERBAND regardless of the input image's map scheme. The comment field of the new image will read the same as the comment field of the input image, except it is append with the band number and the filename from which the band was taken.

The output files are optional arguments, if all or any output files names are not specified, vbandsp3 will use the input filename with a suffix appended to it as the output filenames. The suffixes are determined by the color space model:

NONE or GENERIC:
input_filename.1
input_filename.2
input_filename.3

ntscRGB or genericRGB or cieRGB:
input_filename.R
input_filename.G
input_filename.B

ntscYIQ:
input_filename.Y
input_filename.I
input_filename.Q

ntscCMY:
input_filename.C
input_filename.M
input_filename.Y

HLS: input_filename.H
input_filename.L
input_filename.S

HSV: input_filename.H
input_filename.S
input_filename.V

IHS: input_filename.I
input_filename.H
input_filename.S

cieUVW or cieucsUVW:
input_filename.U
input_filename.V
input_filename.W

cieucsSOW:
input_filename.S
input_filename.O
input_filename.W

cieucsLab:
input_filename.L
input_filename.a
input_filename.b

cieucsLuv:
input_filename.L
input_filename.u
input_filename.v

The -b option determines which data band to extract. The default is 1.

Examples

vbandsp3 -i test

The file test must contain at least 3 data bands, and if it had a color space model = HLS, the output files would be: test.H, test.L, and test.S. If it had no color space model, the output files would be: test.1, test.2 and test.3. In this case bands 0, 1, and 2 would be extracted.

vbandsp3 -i test -o1 bee -o2 junk -o3 yy -b 3

The image "bee" would be band number 3, "junk" would be band number 4 and "yy" would be band number 5. Thus, the image must contain at least 5 data bands.

Command Line Arguments

```
Usage for vbandsp3: Extract 3 Bands from a Multiband Image  (K1)
% vbandsp3
  -i         (infile)  input image filename

  [-b]       (integer) Data band number to extract
```

```
             (b >= 0, default = 0)
    [-o1]    (outfile) output image filename
             (default = {none})
    [-o2]    (outfile) output image filename
             (default = {none})
    [-o3]    (outfile) output image filename
             (default = {none})
```

B.6. vcast — *Upconvert All Inputs to the Highest Input Data Type (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Data Conversion*
Operator:	*Cast Input Types*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vcast will upconvert or upcast each input image to the highest input image data type. Optionally, the user may elect to upcast all images to a specified data type, as long as it is a data type that is equal to or greater than the highest data type of the input images. If the user tries to select a data type that is lower than the highest data type of the input images, an error message will be produced indicating that down casting is not allowed.

There are no restrictions as to the input image data types. The data types can range from BIT to DOUBLE. The input images may contain any combination of sizes or number of data bands. All map information from each input image is transferred to the corresponding output image.

Note that when converting from type BIT to BYTE, the output values will be 0 or 1. This is different from the results one would obtain using the defaults with kconvert. In trying to preserve the integrity of the data, vcast scales the data by 1.

Examples

vcast -i1 image.bit -i2 image.float -o1 image1 -o2 image2

converts both input images to data type FLOAT. The optional data type argument was not selected, so auto convert was used to cast both images to the highest data type of the input images (in this case FLOAT).

vcast -i1 image.bit -i2 image.byte -i3 image.short -o1 img1 -o2 img2 -o3 img3 -t float

converts each input image to data type FLOAT. The optional data type argument was used to upcast all input images to data type FLOAT.

See Also

kconvert

Restrictions

vcast only scales the input data by 1, and will only upconvert the data.

Command Line Arguments

```
Usage for vcast: Upconvert All Inputs to the Highest Input Data Type  (K1)
% vcast
  -i1        (infile)  input image 1
  -i2        (infile)  input image 2
  -o1        (outfile) output image 1
  -o2        (outfile) output image 2

  [-i3]      (infile)  input image 3
             (default = {none})
  [-i4]      (infile)  input image 4
             (default = {none})
  [-i5]      (infile)  input image 5
             (default = {none})
  [-t]       (string)  Select data type [default = Auto Upcast to Highest Input Data Type]
          byte  (choose byte data type),
          int  (choose integer data type),
          complex  (choose complex data type),
          short  (choose short data type),
          float  (choose float data type),
          or double  (choose double data type)
[default: byte]
  [-o3]      (outfile) output image 3
             (default = {none})
  [-o4]      (outfile) output image 4
             (default = {none})
  [-o5]      (outfile) output image 5
             (default = {none})
```

B.7. vclose — *Close Boundaries of an Edge Image (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Segmentation*
Operator:	*Edge Closing*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vclose uses a boundary image and a gradient image as input. Then it tries to close the boundaries by following the maximum gradient between two end points of two contours.

The new closing part is kept if its length is at least "Minimum_Gap_Size" pixels. This parameter is used to prevent the result from being covered by small contours (edges of noisy areas...).

The minimum gradient value "Gradient_Threshold" has to be set up to indicate to the closing process up to what value it can follow a maximum gradient curve to close a boundary.

This Algorithm has been written by:

Professor SERGE CASTAN, Prof CHEN and Dr ZHAO

IRIT, CNRS, URA 1399

118, route de Narbonne 31062 Toulouse FRANCE

Examples

vclose -i1 moon_edge.xv -i2 moon_grad.xv -v 5 -f 0.5 -o closed_boundary.xv

Will use moon_edge.xv and moon_grad.xv as the edges and respectively the gradient representation of an image. Then it will close the unclosed boundaries using a minimum gap size of 5 pixels and a gradient threshold of 0.5

Restrictions

vclose will only work with single band images. Then the first image has to be of data storage type VFF_TYP_1_BYTE. The second input can be anything from VFF_TYP_1_BYTE to VFF_TYP_FLOAT.

Command Line Arguments

```
Usage for vclose: Close Boundaries of an Edge Image  (K1)
% vclose
  -i1        (infile)  Input Boundary Image (result from vdrf, vgef, vsdef ...)
  -i2        (infile)  Input Gradient Image (result from vdrf, vsdef, sobel...)
  -o         (outfile) resulting output image or file

  [-grad]    (integer) Gradient Threshold (used by the closing process)
             (grad >= 0, default = 0)
  [-len]     (integer) Minimum Gap Size In Pixels
             (len >= 0, default = 1)
```

B.8. vcltrans — *Convert Between Tristimulus Color Spaces*

Object Information

Category:	*Khoros 1*	Subcategory:	*Color Transforms*
Operator:	*Linear Convert*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vcltrans performs the linear transformations between YIQ, XYZ, CMY, UVW and RGB color spaces. These transformations were taken from Pratt (Digital Image Processing, 1978). The values in the transformation matrices are based on NTSC RGB standard phosphors, as Pratt describes.

The following transformation matrices are used:

```
RGBtoYIQ = { { 0.299,  0.587,  0.114},
             { 0.596, -0.274, -0.322},
             { 0.211, -0.523,  0.312} };

RGBtoXYZ = { { 0.607,  0.174,  0.201},
             { 0.299,  0.587,  0.114},
             { 0.000,  0.066,  1.117} };

RGBtoUVW = { { 0.405,  0.116,  0.133},
             { 0.299,  0.587,  0.114},
             { 0.145,  0.827,  0.627} };

YIQtoRGB = { { 1.000,  0.956,  0.621},
             { 1.000, -0.272, -0.647},
             { 1.000, -1.106,  1.703} };

YIQtoXYZ = { { 0.967,  0.318,  0.594},
             { 1.000,  0.000,  0.000},
             { 1.173, -1.238,  1.870} };

YIQtoUVW = { { 0.653,  0.208,  0.403},
             { 1.000,  0.000,  0.000},
             { 1.599, -0.780,  0.623} };

XYZtoRGB = { { 1.910, -0.533, -0.288},
             {-0.985,  2.000, -0.028},
             { 0.058, -0.118,  0.896} };

XYZtoYIQ = { { 0.000,  1.000,  0.000},
             { 1.407, -0.842, -0.451},
             { 0.932,  1.189,  0.233} };

XYZtoUVW = { { 0.667,  0.000,  0.000},
             { 0.000,  1.000,  0.000},
             {-0.500,  1.500,  0.500} };

UVWtoRGB = { { 2.432,  0.332, -0.576},
             {-1.519,  2.083, -0.057},
             { 1.440, -2.823,  1.803} };

UVWtoYIQ = { { 0.000,  1.000,  0.000},
             { 1.403,  0.534, -0.907},
             { 1.757, -1.898,  0.470} };

UVWtoXYZ = { { 1.500,  0.000,  0.000},
             { 0.000,  1.000,  0.000},
             { 1.500, -3.000,  2.000} };
```

The input image must be of type FLOAT, and of color space model ntscYIQ, cieXYZ, ntscCMY, cieUVW, ntscRGB or genericRGB. If the image is not in float format, it will be converted automatically, if the conversion is an upward conversion. The input image will be normalized by the value specified by the -n option. The default is 255.

When converting between the different color spaces, care must be taken in regards to the normalization factor. If the image had been scaled by using kscale, then the scale factor used in kscale should be the normalization factor used in this routine. The resulting image should not be normalized by using knormal, if the inverse transformation is desired. The data will be corrupted. NOTE: The output values of some of the transformations will be negative, and some values will be greater than 1. If conversion to BYTE is desired, then a possible solution is:

1. use vbandspt3 to obtain 3 grey level images

2. use kadd to offset the image by the maximum negative value. The offset value will most likely be different for each of the three images.

3. use kstats to get the maximum pixel value in the image and divide that by the maximum desired pixel value. This gives the scale factor. (EX. 3.2/255 = scale factor).

4. now use kconvert to scale (NOT NORMALIZE) by the computed scale factor and convert to BYTE.

5. kinset can be used to create a multi-spectral image.

The reverse of the steps above may be followed to obtain the original image (scale by the inverse of the scale factor).

We really do not know what it means to manipulate this data as explained above, except to make it displayable?

If converting from YIQ, UWV, or XYZ to CMY the routine first converts to RGB then converts to CMY because CMY = 1 - RGB.

Examples

vcltrans -i ball.YIQ.xv -o ball.RGB.xv -t 2 -n 255.0

This will convert the input image, which has a color space model of YIQ, to a color space model of genericRGB. The normalization factor used is 255.

vcltrans -i ball.RGB.xv -o ball.UVW.xv -t 6

This will convert the input image, which has a color space model of RGB, to an image with a color space model of cieUVW. The default normalization factor of 255 is used.

Restrictions

The input image must be of one of the color space models listed above. The data type of the input must be VFF_TYP_FLOAT.

Command Line Arguments

```
Usage for vcltrans: Convert Between Tristimulus Color Spaces
% vcltrans
  -i          (infile)  input image
  -o          (outfile) output resulting image

  [-n]        (float)   specifies normalization factor
              (n > 0.0, default = 255)
  [-t]        (integer) lets user chose color space model
           1  (ntscRGB),
           2  (genericRGB),
           4  (ntscCMY),
           3  (ntscYIQ),
           5  (cieXYZ),
           or 6  (cieUVW)
[default: 1]
```

B.9. vcmtrans — *Linear Color Space Conversion with Specified 3x3 Matrix (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Color Transforms*
Operator:	*Matrix Convert*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vcmtrans will take a multi-spectral input image and output an image of color space model GENERIC by applying a user specified color space conversion matrix.

The input image must be of data storage type FLOAT, and contain three data bands. The color space model for the input image can be set to any type. The input image will be normalized by the value specified by the -n option. The default is 255.

The second input image specifies the transformation matrix. This image must be of data storage type float and must be a 3x3 image. The format of this image must be:

{ 11, 12, 13 }
{ 21, 22, 23 }
{ 31, 32, 33 }

This image can be imported using kasc2val. It is important to remember that the input image will be normalized by the normalization option. The default is 255.0. The conversion is performed as follows:

- Band 1 of the input image is updated by:
  pixel = matrix[0][0]*(t1) + matrix[0][1]*(t2) + matrix[0][2]*(t3)

- Band 2 of the input image is updated by:
  pixel = matrix[1][0]*(t1) + matrix[1][1]*(t2) + matrix[1][2]*(t3)

- Band 3 of the input image is updated by:
  pixel = matrix[2][0]*(t1) + matrix[2][1]*(t2) + matrix[2][2]*(t3)
  where t1 = pixel from band 1
  where t2 = pixel from band 2
  where t3 = pixel from band 3

When converting between the different color spaces, care must be taken in regards to the normalization factor. If the image had been scaled by using kscale, then the scale factor used in kscale should be the normalization factor used in this routine. The resulting image should not be normalized by using knormal, if the inverse transformation is desired. The data will be corrupted. NOTE: The output values of some of the transformations will be negative, and some values will be greater than 1. If conversion to BYTE is desired, then a possible solution is:

1. use vbandspt3 to obtain 3 grey level images

2. use kadd to offset the image by the maximum negative value. The offset value will most likely be different for each of the three images.

3. use kstats to get the maximum pixel value in the image and divide that by the maximum desired pixel value. This gives the scale factor. (EX. 3.2/255 = scale factor).

4. now use kconvert to scale (NOT NORMALIZE) by the computed scale factor and convert to BYTE.

5. kinset can be used to create a multi-spectral image.

The reverse of the steps above may be followed to obtain the original image (scale by the inverse of the scale factor).

We really do not know what it means to manipulate this data as explained above, except to make it displayable?

Restrictions

The input image must be of data storage type FLOAT. The user specified matrix image must have a data storage type for float.

Command Line Arguments

```
Usage for vcmtrans: Linear Color Space Conversion with Specified 3x3 Matrix  (K1)
% vcmtrans
  -i1        (infile)  input image
  -i2        (infile)  input transformation matrix
  -o         (outfile) output resulting image

  [-n]       (float)   specifies normalization factor
             (n > 0.0, default = 255)
```

B.10. vconvert — *Convert Data Storage Type (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Data Conversion*
Operator:	*Convert Type*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

vconvert performs data storage type conversions from any VIFF type to any other, with the exception of the transform types. Scaling or (mutually exclusive) normalization can be performed if requested. The scaling and normalization options are important for controlling the loss of precision when converting from a higher data type to a lower data type.

Output data storage type BYTE is a special case. If scaling is requested, then the input image is scaled, and then clipped to the range [0..255] before output. If normalization is requested, the data is offset adjusted and scaled to map onto the range [0..255] before output.

Normalization for other output types simply scales the data so that the most positive data value is equal to the normalization target. No offset adjustment is performed. If the most positive value in the image is zero then a zero valued image of the desired data type will be produced.

When converting from type BIT to type BYTE, the output values will be 0 or 255 depending on whether the bit corresponding to the input pixel is 0 or 1. For conversions from BIT to any type other than BYTE, the output value receives either 0 or 1 depending on the input bit value. This was done so that an BYTE image would be displayable immediately after conversion from type BIT.

Use of the -b option is important when trying to normalize a multiband or vector image. If -b false is used, then the largest value of any vector **component** in the image is used to compute the normalization multiplier. If -b true is used, then a different normalization multiplier is used for each band, as dictated by the data in that band.

Examples

vconvert -i image:moon -s 3.14 -t float -o blah.xv multiplies "image:moon" by pi then outputs it as a VFF_TYP_FLOAT image.

See Also

kconvert, knormal

Restrictions

This routine was ported to the Retro toolbox only because other Retro operators needed the lvconvert library call. Use the new kconvert or knormal programs instead of this program!

There needs to be a way to specify the conversion from a COMPLEX type to a non-COMPLEX type to be either the real part or the magnitude (the current version does the real part only).

Command Line Arguments

```
Usage for vconvert: Convert Data Storage Type  (K1)
% vconvert
  -i         (infile)  input image
  -o         (outfile) resulting output image

  [-t]       (string)  lets user choose output data type
          bit  (choose bit data type),
          int  (choose integer data type),
          double  (choose double data type),
          byte  (choose byte data type),
          float  (choose float data type),
          dcomplex  (choose double data type),
          short  (choose short data type),
          or complex  (choose complex data type)
[default: int]
  [-b]       (boolean) if true, bands are processed independently
             (default = false)

  [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
    [-n] (float) specifies normalization value
         (n > 0.0, default = 255)
    [-s] (float) specifies scale factor
         (unbounded, default = 1)
```

B.11. vconvolve — *Convolution of an Image and a Kernel (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Linear Operators*
Operator:	*Convolution*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

NOTE: Program will NOT execute! vconvolve functionality is replaced by DATAMANIP::klinearop. To reproduce the vconvolve default settings, run klinearop with the following additional command line arguments. (the output data type may be different)

% klinearop -w -h -l -size 0 -real 0 -imag 0 *vconvolve* performs a two dimensional convolution of an image a specified kernel image.

If either of the files is of data storage type FLOAT, the output will be in that format, otherwise the output will be 4_BYTE (integer).

The second input image should be smaller than the first input image as it is treated as the filter kernel or weights.

Edge problems are handled by truncating the part of the kernel that is off of the image array.

Examples

vconvolve -i1 image:moon -i2 laplace3x3.xv -o result.xv

Convolves the laplacian3x3 kernel with "image:moon" and leaves the result in "result.xv". The data type of the output image will be VFF_TYP_4_BYTE (32-bit integer).

See Also

DATAMANIP::klinearop DATAMANIP::kfft

Restrictions

The 5D klinearop program in the datamanip toolbox replaces this 2D convolution program

vconvolve will not operate on VFF_TYP_COMPLEX images. Accumulator error may occur when convolving two large images.

Command Line Arguments

```
Usage for vconvolve: Convolution of an Image and a Kernel  (K1)
% vconvolve
  -i1        (infile)  first input image
  -i2        (infile)  second input image (filter kernel)
  -o         (outfile) resulting output image
```

B.12. vcost — *Compute Cost (Surface Arc Length) for Image (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Surface*
Operator:	*Compute Cost*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vcost computes the surface arc length from a given pixel location to every other pixel in the image. The surface distance from a particular pixel to the reference pixel location is stored in the corresponding location in the output image. Image pixels are resampled at 1/10 of their spacing using bilinear interpolation for use in forming the surface profile between any two points. This provides an surface distance accurate to around 1% for relatively smooth images. An image with very sharp transitions or other high frequencies will produce less accurate results.

If the mask image is used, then only those pixels that have a non-zero value at the corresponding pixel in the mask will be evaluated for surface distance. All other output pixels are set to -1.

The output is of type float.

Examples

% vcost -i surface -o dist -x 16 -y 21

Restrictions

lvcost works on all data types.

Command Line Arguments

```
Usage for vcost: Compute Cost (Surface Arc Length) for Image  (K1)
% vcost
  -i          (infile)  input image
  -o          (outfile) output image

  [-m]        (infile)  mask image
              (default = {none})
  [-x]        (integer) X coordinate of reference pixel
              (unbounded, default = 0)
  [-y]        (integer) Y coordinate of reference pixel
              (unbounded, default = 0)
```

B.13. vdetilt — *Remove Illumination Gradient by Subtracting Best-Fit Plane (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Surface*
Operator:	*Remove Tilt*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vdetilt computes the best-fit plane for an image, and then subtracts that plane from the image to produce the output. This is very handy for correcting illumination gradients in a poorly digitized image. The "remove mean" option also removes the mean from the image, giving it zero-mean statistics. If "remove mean" is set to yes (default), then remove mean from image.

The optional mask image must be of the same data type as the input images, and is used to gate the operation. A non-zero mask pixel enables the output pixel to contain the adjusted value. A zero mask pixel just transfers the value of the corresponding pixel in the first input image to the output pixel.

See Also
vgettilt, lvgettilt, vtilt, lvtilt

Restrictions
vdetilt will not work on BIT, transform or COMPLEX data storage types.

Command Line Arguments

```
Usage for vdetilt: Remove Illumination Gradient by Subtracting Best-Fit Plane  (K1)
% vdetilt
  -i         (infile)  input image
  -o         (outfile) resulting detilted image

  [-m]       (infile)  find tilt  operation masking image
             (default = {none})
  [-p]       (boolean) if yes, remove mean from image
             (default = true)
```

B.14. vdistance — *Compute Distance from the Boundary in a Two-Valued Image (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Segmentation*
Operator:	*Border Distance*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description
vdistance Computes the distance from the nearest boundary point.

The **distance transform** uses the following iterative algorithm :

U(m,n) : **original image**. This image contains only two gray levels, 0 for the background, and any other value for the objects.

Uk(i,j): **image after k iterations**.

U0(m,n)= U(m,n), and for k= 1, 2, 3, ...

Uk(m,n)= U0(m,n)+min{Uk-1(i,j);((i,j):dist(m,n;i,j)<= 1)},

dist(m,n;i,j) is the distance between (m,n) and (i,j).

dist(i,j-1;i,j) = dist(i,j+1;i,j) =dist(i-1,j;i,j) =dist(i+1,j;i,j) = 1.

dist(i-1,j-1;i,j) = dist(i+1,j-1;i,j) =dist(i-1,j+1;i,j) =dist(i+1,j+1;i,j) = 2.

The transform is completed when k equals the maximum thickness of the region. That is, when Uk+1 equals Uk.

The distance image can be used to **extract edges** or the **medial axis** (form of skeleton).

Set of functions that can be used with vdistance:

vmediaxis Customizes the medial axis from the distance image.

kthresh Generate a binary image.

ADVICE: For this routine, the image must be a byte binary image, with the gray-level value 0 for the background.
It is advisable to perform a threshold operation prior to using vdistance.

Examples

vdistance -i image1 -o image2
Computes the distance from the nearest boundary point.
For this routine, the image has to be a byte binary image, with the gray-level value 0 for the background. So it is strongly advisable to perform thresholding before running vdistance.

See Also

kclip

Restrictions

vdistance works only on BYTE binary images which background value is zero.

Command Line Arguments

```
Usage for vdistance: Compute Distance from the Boundary in a Two-Valued Image  (K1)
% vdistance
  -i          (infile)  input image
  -o          (outfile) output image
```

B.15. vdrf — *Optimal Difference Recursive Filter for Edge Detection (K1)*

Object Information

Category: *Khoros 1* Subcategory: *Spatial Filters*

Operator: *DRF Edge Extract* Object Type: *kroutine*
In Cantata: *Yes*

Description

This Algorithm has been written by the team of:

Professor SERGE CASTAN

IRIT, URA 1399

118, route de Narbonne 31062 Toulouse FRANCE

OPTIMAL FILTER(ISEF) FOR EDGE DETECTION

PRINCIPLE :

I. Introduction

Edge detection is one of the most important subjects in image processing, which finds wide applications in the pattern recognition, the scene analysis and the 3-D vision, because the edges correspond in general to the important changes of physical or geometrical properties of objects in the scene and they are widely used as primitives in pattern recognition, image matching etc.

The edges coincide, generally speaking, with grey level transition, so they can be detected by maxima of gradient or the zero-crossing of the second derivatives calculated by some differential operators. Because the differential operators are sensitive to noise, a preprocessing such as smoothing is often necessary to eliminate the noise.

One well-known smoothing filter is the Gaussian filter and therefore the edges can be detected by a Laplacian-Gaussian filter. But there is an essential difficulty of the Laplacian-Gaussian filter which is the contradiction between the smoothing effect and the precision of edge localization. To overcome this difficulty, We proposed the optimal linear filter based on one step model (a step edge and the white noise) and the multi-edge model [9][10][11]. This optimal smoothing filter is a symmetric exponential filter of infinitely large window size and can be realized by a very simple recursive algorithm.

It is proved that the band limited Laplacian of an input image filtered by this filter can be computed from the Difference between the input and the output of this Recursive Filter (DRF). The edges detected by DRF method are less noisy and with a much better precision of localization.

II. The Difference Operators for Symmetric Exponential Filter

A normalized symmetric exponential filter on 1-D can be written :

$$fL(x)=C*a0*(1-a0)**|x|=f1(x)\#f2(x)=C*(f1(x)+f2(x)-a0*d(x)) \quad (1)$$

where:

$$f1(x) = \{0 \text{ if } x<0 \text{ and } a0*(1-a0)**x \quad \text{if } x >= 0 \quad (2)$$

$$f2(x) = \{0 \text{ if } x>0 \text{ and } a0*(1-a0)**-x \quad \text{if } x <= 0 \quad (2)$$

Because the exponential function is separable, we can write out 2-D exponential filter:

$$f(x,y) = fL(x)*fL(y) \qquad (5)$$

III. The Recursive Algorithm for realizing the These Directional Derivative Operators of Symmetric Exponential Filter

The exponential filter is an IIR filter corresponding to an infinite window size, so we should realize the functions $f1(x)$ and $f2(x)$ (see formula (2)) by a recursive algorithm.

Supposing $I(x,y)$ is the input image, $I1(x,y)=I(x,y)\#f1(x)$ and $I2(x,y)=I(x,y)\#f2(x)$, we have the recursive algorithm :

$$I1(x,y) = I1(x-1,y) + a0*(I(x,y) - I1(x-1,y)) \quad (10)$$

$$I2(x,y) = I2(x+1,y) + a0*(I(x,y) - I2(x+1,y))$$

From the equations (1),(6),(7),(8) and (9), the band-limited first and second directional derivative of input image can be calculated by the recursive algorithm f1 and f2 as follows

IV. Edges Detection

The band-limited Laplacian of an input image filtered by this filter can be calculated from the Difference between the input and the output of this Recursive Filter (DRF).

1. Edges from the maxima of Difference

The Difference magnitude image is then non maxima suppressed in the gradient direction and thresholded with hysteresis, i.e. if the entire segment of the contour lies above a low threshold T1, and at least one of part of which is above a high threshold T2, that contour is output. The non maxima suppression scheme requires three points, one of which will be the current point, and the other two should be estimated of the gradient magnitude at points displaced from the current point by vector normal to the edge direction.

V. Comparison of Performance of the Filters

Filtering is a problem of estimation from noisy signal, and edge detection is a problem of estimating the position of maximal local signal change. Up to now, many works are done for edges detection in image, and different filters are proposed, for example, Gaussian filter, Canny filter, exponential filter, Deriche filter etc...

We appreciate the performance of the filters as follows :

(1) Precision of edge localization

According to our analysis [10], we can calculate the average localization error xe for Gaussian filter, Canny filter [15], Deriche filter [16] and the exponential filter :

xeG = (4*(2*e*3.14)**0.5)/a

xec = 0.81/a

xeD = 4*exp(-1)/a = 1.47/ a

xeE = 0

i.e. xeG > xeD > xec > xeE = 0.

So, we can see that the exponential filter localizes edge points with the best precision.

(2) Signal/Noise ratio on the edge point detected

Because xe is the average estimation for the position of the edge point detected, we propose to calculate Signal/Noise ratio (Eq.(7)) at the point xe.

And the signal/noise ratio for the Gaussian filter, Canny filter, Deriche filter and the exponential filter is :

SNRG = 2*s*exp(-32*s)/ ((3.14)**0.5)

SNRc = 0.39/ a

SNRD = 0.64/ a

SNRE =1/ a

i.e. SNRE > SNRD > SNRc > SNRG.

Then, we see that the exponential filter has the best noise eliminating effect among the above four.

(3) Complexity of calculation

For the complexity of calculation, we only tell the difference from exponential filter and Deriche filter [16], because they are implemented by recursive algorithms which have a simpler calculation.

Because ISEF can be realized by first order recursive filter, the ISEF algorithms are much simpler than that of Deriche filter. Besides, the ISEF algorithms can be implemented independently to every line and every column, it can be easily realized by a parallel system.

According to the analysis results above, the ISEF filter is superior to the others at the 3 principal aspects of the performance of the filter.

VI. Conclusion

The symmetric exponential filter of an infinite large window size is an optimal linear filter deduced from one step edge model and the multi-edge model, now we further prove that the symmetric exponential filter is the optimal edge detection filter in the criteria of the signal to noise ratio, the localization precision and unique maximum. Obviously, the real images will be still more complicated than these models, however DRF method has already provided good results for different type of images. The results obtained through the two new methods further (ie. SDEF and GEF : vsdef and vgef) show the superior performance of this filter. The theoretical analysis for the performance of the filters shows also that the exponential filter is superior to the other current filters.

Examples

vdrf -i cross.xv -o output.xv

will compute a DRF edge extraction (difference recursive filter) on the image cross.xv and write the result in output.xv.

For the DRF edge extraction the filter parameters values are generally a1 = a2 = 0.3

The size of the window has to be odd and 5 is generally a good value.

The hysteresis Thresholds T1 and T2 can be set to 12 and 15.

See Also

lvdrf(3)

Restrictions

Works only with single band byte images.

Note that this routine was converted directly from Khoros 1.0. Due to the mixing of unsigned versus signed char, and the difference in the rules that resolve the mixing of these data types between ANSI C and the original K&R C that the code was written in, the results of running this program on an image will not be the same as with vdrf in Khoros 1.0. There is really no way to resolve which version is more correct, but the ANSI C version should at least be more consistent across machine architectures.

References

[1] W.K. PRATT, Digital Image Processing, New York, 1978.

[2] J. PREWITT, Object Enhancement and Extraction,Picture Processing and Psychopictories, Etd. by B. Lipkin and A. Rosenfeld, New York, pp.75-149, 1970.

[3] M. HUECKEL, An Operator Which Locates Edges in Digitized Pictures. J.A.C.M., Vol. 18, pp 113-125, 1971.

[4] R.O. DUDA and P.E. HART, Pattern Classification and Scene Analysis. Wiley, New York, 1973.

[5] R. HARALICK,Edge and Region Analysis for Digital Image Data. C.G.I.P., Vol. 12, pp 60-73, 1980.

[6] R.HARALICK and L.WATSON, A Facet Model for Image Data. C.G.I.P., Vol. 15, pp 113-129, 1981.

[8] D. MARR and E.C. HILDRETH, Theory of Edge Detection. Proc. R. Soc. Lond. B, Vol. 207, pp 187-217, 1980.

[9] J. SHEN and S. CASTAN, Un nouvel algorithme de detection de contours, proceedings of 5th Conf. on P.R.&.A.I. (in French), Grenoble, 1985.

[10] J. SHEN and S. CASTAN, An Optimal Linear Operator for Edge Detection. Proc. CVPR'86, Miami,1986.

[11] J. SHEN and S. CASTAN, Edge Detection Based on Multi-Edge Models.Proc. SPIE'87, Cannes, 1987.

[12] J. SHEN and S. CASTAN, Further Results on DRF Method for Edge Detection. 9th I.C.P.R., ROME, 1988.

[13] V.TORRE and T.A.POGGIO, On Edge Detection IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. Pami-8, N 2, March 1986.

[14] J.S.CHEN and G.MEDIONI, Detection, Localization, and Estimation of Edges. IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. 11, N 2, February 1989.

[15] J.F.CANNY, Finding Edges And Lines in Images. MIT Technical Report N 720, 1983.

[16] R. DERICHE, Optimal Edge Detection Using Recursive Filtering. In proc. First International Conference on Computer Vision, London, June 8-12 1987.

Command Line Arguments

```
Usage for vdrf: Optimal Difference Recursive Filter for Edge Detection  (K1)
% vdrf
  -i         (infile)  input image
  -o         (outfile) output image

  [-a1]      (float)   a1 Exponential Filter Parameter for DRF
             (0 <= a1 <= 1, default = 0.3)
  [-a2]      (float)   a2 Exponential Filter Parameter for DRF
             (0 <= a2 <= 1, default = 0.3)
  [-w]       (integer) Window size for calculating the adaptive gradient
             (0 <= w <= 11, default = 5)
  [-t1]      (integer) 1st Hysteresis Threshold
             (0 <= t1 <= 255, default = 12)
  [-t2]      (integer) 2nd Hysteresis Threshold
             (0 <= t2 <= 255, default = 15)
```

```
  [-l]       (integer) Minimum Pixel Number in a Segment
             (0 <= l <= 100000, default = 10)
```

B.16. vdyth — *Perform Dynamic Thresholding on Image (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Segmentation*
Operator:	*Dynamic Threshold*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vdyth performs dynamic thresholding on an image by either using the median value or mean value of a window (mask) as the threshold value. The window is specified by its width argument, w, and its height argument, h. Thresholding is performed by moving the window over the image, and at each new position a new threshold value is determined for the pixel currently at the center of the window.

Examples

vdyth -i input.image -o output.image -w 3 -h 3 -v 11 -t 0

This command performs a dynamic threshold on input.image and stores the result in output.image. The size of the window for the operation is 3 by 3. The value of non-zero pixels in the output will be 11, and the median is used to determine the threshold.

Restrictions

vdyth supports only BYTE data storage type images.

Command Line Arguments

```
Usage for vdyth: Perform Dynamic Thresholding on Image  (K1)
% vdyth
  -i         (infile)  input image
  -o         (outfile) resulting output image

  [-w]       (integer) window width argument
             (1 <= w <= 100, default = 3)
  [-h]       (integer) window height argument
             (1 <= h <= 100, default = 3)
  [-v]       (integer) non zero output pixel value
             (0 <= v <= 255, default = 255)
  [-t]       (boolean) Use Median as Threshold [False] or Mean [True]
             (default = false)
```

B.17. venhance — *Enhance Image Using the Local Standard Deviation & Mean (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Histogram Operators*
Operator:	*Local Enhance*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

venhance uses a local transformation based on the mean and variance of the intensity maps and the intensity of the input image f(x,y) to create a new image g(x,y) by performing the following transformation at each pixel location (x,y):

g(x,y) = A(x,y).[f(x,y) -m(x,y)] + m(x,y),

where

A(x,y) = k . M / sigma(x,y)   0 < k < 1

where m(x,y) and sigma(x,y) are the gray-level mean and standard deviation computed in a neighborhood centered at (x,y), M is the global mean of f(x,y), and *k* is a constant in the range indicated above.

Examples

venhance -i input.image -o output.image -w 5 -h 5 -k .5 -v 2

This command performs image enhancement on input.image and stores the result in output.image. The size of the window used is 5 by 5. The tuning factor is .5 and the minimum allowable standard deviation is 2.

See Also

Reference: Digital Image Processing (2nd Edition) by Gonzalez and Wintz.

Restrictions

venhance only operates on byte type images.

References

Digital Image Processing (2nd Edition) by Gonzalez and Wintz.

Command Line Arguments

```
Usage for venhance: Enhance Image Using the Local Standard Deviation & Mean  (K1)
% venhance
  -i         (infile)  input image filename
  -o         (outfile) output image filename

  [-w]       (integer) window width of local area
```

```
            (1 <= w <= 100, default = 3)
[-h]        (integer) window height of local area
            (1 <= h <= 100, default = 3)
[-k]        (float)   specifies scale factor
            (0 <= k <= 1, default = 0.5)
[-v]        (float)   specifies minimum standard deviation
            (0 <= v <= 2000, default = 1)
```

B.18. vextract — *Extract Rectangular Region from Object (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Size Operators*
Operator:	*Extract Region*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

NOTE: Program will NOT execute! vextract functionality is replaced by the DATAMANIP::kextract operator. To reproduce the vextract default settings, run kextract with the following command line arguments.

kextract -wsize 128 -hsize 128 -woff 0 -hoff 0

vextract copies that area of the input image which is w pixels wide and h pixels high with the upper left corner at (x,y) into a new image, called a subimage. The subimage can then be treated just like an normal image and can be reinserted back into the original or any other image of the same data type.

If the subimage specification extends beyond the image boundary, the largest possible subimage is returned. If the subimage is larger than the original image, the routine will fail.

Vextract allows a coordinate input image to specify the region to extract rather than specifying the region by the -x, -y, -h and -w arguments. If the -i1 argument (input coordinate image) is specified then none of the specific coordinate values (-x, -y, -w -h arguments) can be specified. The coordinate input image must contain at least 2 columns and 3 rows. The following explains how the coordinate input image is used to extract a sub-region. The first 2 data points in the image are the x any y offset values. These specify the -x and -y arguments. To find the width of the subimage, the first value in the second row is subtracted from the first value of the first row. To find the height of the subimage, the second value in the third row is subtracted from the second value of the first row. Basically the coordinate image should be specified as (but is not limited to) an image with three rows and two columns where each row specifies the (x,y) location of a point. Then the first row specifies the (x,y) location of the upper left hand corner, the second row specifies the (x,y) location of the upper right hand corner and the third point specifies the lower left hand corner.

The output image is the same data type as the input image. An image with only the color map and header data is obtained if none of the options are used.

The input image may be single or multiband, and the resulting output image will be of the same type and same number of bands.

Examples

vextract -i ball.xv -o ball_sub.xv -x 64 -y 64 -w 128 -h 128

This extracts a subimage from ball.xv beginning at coordinates (64, 64), and the resulting output image is of dimension 128 by 128.

See Also

DATAMANIP::kextract

Command Line Arguments

```
Usage for vextract: Extract Rectangular Region from Object  (K1)
% vextract
  -i         (infile)  input image
  -o         (outfile) resulting output image

  [-i1]      (infile)  input coordinate image
             (default = {none})
  [-x]       (integer) specifies upper left x coordinate of subimage
             (x >= 0, default = 0)
  [-y]       (integer) specifies upper left y coordinate of subimage
             (y >= 0, default = 0)
  [-w]       (integer) specifies the subimage width in pixels
             (w > 0, default = 128)
  [-h]       (integer) specifies the subimage height in pixels
             (h > 0, default = 128)
```

B.19. vfht — *Compute 2-D Fast Hadamard Transform (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Transforms*
Operator:	*Hadamard*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vfht performs a 2-D Fast Hadamard Transform on an image. The Hadamard Transform uses a series of pulses as its basis functions. This program uses the successive doubling algorithm and the separability property to implement a Fast Transform. The output coefficients are, by definition, in sequency order.

The output data may be scaled by 1/N, or not scaled at all. The default is to not scale the data at all. The only difference between a forward and a reverse Hadamard Transform is the scaling factor, so this routine functions as both a forward and reverse transform provided that one of the transforms (forward OR reverse) is set to scale by 1/N and the other performs no scaling.

The Following references were used in creating this program :

1. "Digital Image Processing, Second Edition"
   Rafael C. Gonzalez

Paul Wintz
Addison-Wesley Publishing Company, Reading Massachusetts, 1987

2. "The C User's Journal", July 1989, "The Fast Walsh Transform", p101 .br
Ian AshDown

Examples

vfht -i input.image -o hadamard.domain.image

This command performs a fast Hadamard transform on the image contained in the viff file "input.image" and returns the output in another viff file called "hadamard.domain.image".

If your input image had dimensions 4 row by 4 columns, and the image had a constant value as shown below

```
input image = { 1, 1, 1, 1,
                1, 1, 1, 1,
                1, 1, 1, 1,
                1, 1, 1, 1 }
```

The output would be this

```
output image = { 1, 0, 0, 0,
                 0, 0, 0, 0,
                 0, 0, 0, 0,
                 0, 0, 0, 0 }
```

Because the input image was simply the first of the Hadamard basis functions with magnitude = 1.

See Also

dfht(1)

Restrictions

The input data type may be byte (VFF_TYP_1_BYTE), short (VFF_TYP_2_BYTE), int (VFF_TYP_4_BYTE) or float (VFF_TYP_FLOAT), but the output coefficients will always be float. The dimensions of the input image must both be non-zero powers of two. Images with maps are not allowed. The input viff file must contain only one image with a single band. Data with explicit locations is not accepted.

Command Line Arguments

```
Usage for vfht: Compute 2-D Fast Hadamard Transform  (K1)
% vfht
  -i         (infile)  input image (viff file)
  -o         (outfile) output image (viff file)

  [-s]       (boolean) scale output by 1/N (0 = no scale, 1 = scale)
             (default = false)
```

B.20. vfractal — *Fractal Feature Extraction (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Feature Extraction*
Operator:	*Fractal Analysis*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vfractal performs a fractal feature extraction of an image via calculation of the fractal dimension "D". The fractal dimension is computed based on a calculation of the Hurst coefficient within a selected window size or "fractal region" of the image. The Hurst coefficient is a measure of the slope of the line relating the window size vs average range of image intensity. The Hurst coefficient is related to the fractal dimension as follows,

$$D = 3 - H$$

where,

D - is the resulting fractal dimension
H - is the computed Hurst coefficient
3 - constant related to the Euclidean geometry

The fractal dimension operation is computed, using overlapping windows, over the entire image. A range of window sizes up to one third of the size of the fractal region are used to compute the fractal dimension. The center pixel in each fractal region is updated based on the Hurst coefficient and subsequent fractal dimension calculation for that region.

The input image, which must be of data storage type FLOAT, corresponds to the -i argument. The output image, which corresponds to the -o argument, is stored as data type FLOAT. The resulting output image will be a single band image, and will have a border consisting of pixels of value 0, since the window operation does not calculate values for border pixels.

The size of the window for the "fractal region" is specified by the -r region size argument. The region size argument cannot be an even number. This means that the region dimensions must be odd (ie. 15x15, 17x17, etc.), since the center pixel is replaced with the computed value of the window data. The default region size is 15x15.

The initial size of the sliding window in the fractal region is specified by the -w initial sliding window size argument. The size of the sliding window cannot be less than 2 and cannot exceed 2 minus the upper window size. The upper window size is determined by the region size, and cannot exceed one-third of the region size. The size of the lower window cannot exceed 2 minus the upper window size, because at least two points are needed to calculate the slope of the best fit line.

The step size specifies the increment to use when incrementing the sliding window size from the initial size (-w argument) to the upper window size (determined from the region size). The step size cannot exceed a value that would prevent at least two points from being used to calculate the slope of the best fit line. The step size is specified by the -s step size argument.

Each pixel in the image is updated with a computed value except the border pixels. A 15x15 window will result in a border of seven pixels, a 17x17 window will result in a border of eight pixels, etc.

The resulting feature vector will contain one band of data, which contains the computed fractal dimension "D" of the image intensity data for the particular region size selected. Note that different results may be obtained by varying the region size and initial size of the sliding window.

Comments - To calculate the overall fractal dimension of an image, set the region size to be the same size as the image (Note: region size must be odd), and use an initial sliding window size about one-fourth to one-third of the region size. This will result in one value for the fractal dimension of the image, which can be obtained from running kstats.

When running this algorithm on a large image using a wide range of sliding window sizes, it may be helpful to use a generous step size to speed up the processing of the image. For example, if a 256 by 256 image is being processed with a region size of 65 and an initial sliding window size of 3, it may be helpful to select a step size of about 6. The following window sizes would be used in the calculation of the Hurst coefficient (slope of best fit line): window sizes --> 3 9 15 21

If finer resolution is needed, a smaller step size should be used at the expense of processing time.

To view the resulting fractal dimension image in 3-D, use xprism to plot the image as a surface plot.

The input image MUST be of data storage type FLOAT.

The output image is of data storage type FLOAT.

Examples

vfractal -i input_image.xv -o output_image -r 65 -w 3 -s 6

the selected window size of the fractal region is 65 x 65, which will result in a border width of 32 pixels in the output image. The initial starting size of the sliding window is 3 x 3, and will range in size up to one-third of the region size, or 21 x 21 in this case. The step size is 6, which will result in window sizes of 3, 9, 15, and 21 to be used in the calculation of the Hurst coefficient. Note that a tradeoff between resolution of the best fit line and processing time was made by using a step size of 6.

See Also

vgfractal(1)

Restrictions

vfractal will only operate on images of data storage type FLOAT.
The output image is of data storage type FLOAT.

Command Line Arguments

```
Usage for vfractal: Fractal Feature Extraction  (K1)
% vfractal
  -i         (infile)  input image filename
  -o         (outfile) output image filename
```

```
[-r]       (integer) region size argument
           (r > 0, default = 15)
[-w]       (integer) initial window size argument
           (w > 0, default = 2)
[-s]       (integer) step size argument
           (s > 0, default = 1)
```

B.21. vgamth — *Generate a Binary Image by Thresholding a Color Image (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Segmentation*
Operator:	*Color Threshold*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vgamth generates a binary image by thresholding a pseudo color input image. The result image is of data type BYTE, and the input image must also be of type BYTE.

The output pixel is given the specified value if: 1) only the lower value option is given and the pixel value is larger then the lower threshold, 2) only the upper value option is given and the pixel value is less than the upper threshold, 3) the upper value and lower value options are given and the pixel value is between the upper and lower threshold, or 4) same as 3 but both thresholds are equal and the pixel value is same as the threshold level. Otherwise, the output pixel is given a zero value.

The input image must contain a map with three columns The thresholding is actually performed on the maps; thus the input image must have a map scheme of FORCE, and the map data type must be BYTE.

By choosing the appropriate limits is it possible to identify areas in an image that are above or below a certain value, or to isolate certain bands of intensities in each color map. There are many different combinations that can be used to produce the results required.

Note that this type of image is NOT a Khoros BIT image! The output image is of the data type BYTE, but the only pixel values are 0 and non-zero.

Examples

vgamth -i input.image -o output.image -b 125 -c 75 -d 125 -e 75 -f 125 -g 75 -v 100

This command performs a threshold on input.image and stores the result in output.image. All pixel values between 75 and 125 are given the value 100 in the output image. All pixel values that do not fall within this range are give the value 0.

Restrictions

vgamth will only operate on 1 band images of data storage type BYTE, with the map enable FORCE, and map data type BYTE.

Command Line Arguments

```
Usage for vgamth: Generate a Binary Image by Thresholding a Color Image  (K1)
% vgamth
  -i          (infile)  input image
  -o          (outfile) resulting image

  [-b]        (integer) high threshold for map column 1
              (0 <= b <= 255, default = 128)
  [-c]        (integer) low threshold for map column 1
              (0 <= c <= 255, default = 128)
  [-d]        (integer) high threshold for map column 2
              (0 <= d <= 255, default = 128)
  [-e]        (integer) low threshold for map column 2
              (0 <= e <= 255, default = 128)
  [-f]        (integer) high threshold for map column 3
              (0 <= f <= 255, default = 128)
  [-g]        (integer) low threshold for map column 3
              (0 <= g <= 255, default = 128)
  [-v]        (integer) non-zero output value
              (0 <= v <= 255, default = 128)
```

B.22. vgbox — *Create Image of Parallelogram Projected onto CCD Sensor (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Generate Data*
Operator:	*Box Projection*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

This program create a grey level image file filled with a representation of a parallelogram (box) projected on the CCD sensor. It calculates the grey level of each pixel from the intersection of the theoretical parallelogram and the rectangular active area of each sensor-pixel. The sensor is made of pixels supposed to be all identical, composed of an active rectangular area in a rectangular cell.

14 optional parameters can be defined:

For the sensor:

its number of columns (=rows_size) and number of rows (=cols_size), the size of each pixel active area, the position of active area on the whole pixel area.

For the parallelogram:

X and Y sizes, its position (horizontal and vertical shifts), its rotation (in degrees). its shape (parallelogram). The grey levels of full lighted pixels and full dark pixels.

Speed:

about 0.5 ms /pixel on a Sun 3. about 0.04 ms /pixel on a Sun 4 sparc station IPC.

Warning:

The X and Y sizes of the parallelogram is the Euclidean distances between the parallel lines.

Update:

August 1991 : new optional parameter for parallelogram

The trigger input can be used to cause *vgbox* to re-execute when used inside cantata. No image is read from this input.

Examples

vgbox -o image.xv

Will create an xv image "image.xv" with defaults parameters.

vgbox -o box1.xv -r 64 -c 64 -px_w 0.7 -px_h 0.8 -px_x_off 0.1 -px_y_off 0.15

Will create a square xv image "box1.xv" 64*64 pixels with the default box. Each CCD pixel have an active area of 70% * 80% and is shift 10% of a pixel width on the right and 15% downward of a pixel height from the top-left corner of its pixel cell.

vgbox -o box2.xv -r 64 -c 32 -w 10.5 -h 40.3 -im_x_off 20.4 -im_y_off -10.6 -angle 5.1

Will create an xv image "box2.xv" 64 rows by 32 columns with a rectangular box of 10.5 * 40.3 pixels. The box is 20.4 on the right and 10.6 upward, the rotation is 5.1 degrees clockwise..

vgbox -o image.xv -corner_angl 60

Will create an xv image "image.xv" not a rectangular box but a parallelogram.

See Also

lvgbox

Restrictions

Pixel width + pixel offset must be equal or less than 1.000.

The CCD cells are supposed square in order to obtain the asked angles.

For a parallelepipedic box, the width and height sizes of the box are the Euclidean distances between the parallel lines.

Command Line Arguments

```
Usage for vgbox: Create Image of Parallelogram Projected onto CCD Sensor  (K1)
% vgbox
  -o             (outfile) resulting output image

  [-trigger]     (infile)  trigger input
                 (default = {none})
  [-r]           (integer) Number of Rows (Height in pixels) of the CCD sensor
```

```
                 (r >= 0, default = 512)
  [-c]           (integer) Number of Columns (Width in pixels) of the CCD sensor
                 (c >= 0, default = 512)
  [-px_h]        (float)   Active Pixel Height (vertical)
                 (0 <= px_h <= 1, default = 1)
  [-px_w]        (float)   Active Pixel Width (horizontal)
                 (0 <= px_w <= 1, default = 1)
  [-px_y_off]    (float)   Active Pixel offset from the top  of the cell
                 (0 <= px_y_off <= 1, default = 0)
  [-px_x_off]    (float)   Active Pixel offset from the left of the cell
                 (0 <= px_x_off <= 1, default = 0)
  [-h]           (float)   Height of the parallelogram (box) (in pixel unit)
                 (h >= 0.0, default = 40.3)
  [-w]           (float)   Width  of the parallelogram (box) (in pixel unit)
                 (w > 0.0, default = 20.6)
  [-im_y_off]    (float)   Downward offset of box (0 is CCD center = box center)
                 (unbounded, default = 0)
  [-im_x_off]    (float)   Right    offset of box (0 is CCD center = box center)
                 (unbounded, default = 0)
  [-angle]       (float)   Box rotation (clockwise degrees, center = box center)
                 (unbounded, default = 15)
  [-corner_angl] (float)   the value of the 1st corner angle (degrees)
                 (1e-06 <= corner_angl <= 179.999, default = 90)
  [-g_o]         (integer) Grey level of the background pixels outside of box
                 (0 <= g_o <= 255, default = 10)
  [-g_i]         (integer) Grey level of the lighted pixels inside of box
                 (0 <= g_i <= 255, default = 200)
```

B.23. vgcirc — *Create Image of a Filled Circle (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Generate Data*
Operator:	*Circle Image*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vgcirc creates an image of a filled circle. The background intensity is set by the -b argument. The foreground or circle intensity is set by the -f argument. The circle can be placed anywhere, and is NOT required to lie completely within the image boundaries.

The position of the center of the circle and its diameter can all be specified as floating point numbers, allowing sub-pixel positioning of the circle.

The trigger input can be used to cause *vgcirc* to re-execute when used inside cantata. No image is read from this input.

Examples

vgcirc -r 32 -c 32 -b 3.5 -f 107.5 -x 32 -y 16 -d 32 -t float

Creates a 32x32 image of type FLOAT, with the background level at 3.5 and the foreground at 107.5. The circle lies in this case halfway off the image to the right. The diameter of the circle is 32 pixels.

Restrictions

vgcirc will generate only BYTE or FLOAT images.

Command Line Arguments

```
Usage for vgcirc: Create Image of a Filled Circle  (K1)
% vgcirc
  -o         (outfile) output image

  [-trigger] (infile)  trigger input
             (default = {none})
  [-r]       (integer) Number of rows in output image
             (r > 0, default = 512)
  [-c]       (integer) Number of columns in output image
             (c > 0, default = 512)
  [-d]       (float)   Diameter of circle (in pixels)
             (d >= 0.0, default = 256)
  [-x]       (float)   X coordinate of center of circle
             (unbounded, default = 256)
  [-y]       (float)   Y coordinate of center of circle
             (unbounded, default = 256)
  [-b]       (float)   Background level
             (unbounded, default = 0)
  [-f]       (float)   Foreground level
             (unbounded, default = 255)
  [-t]       (string)  Selects data type of output image
          byte  (choose Byte data storage type),
          or float  (choose Float data storage type)
[default: byte]
```

B.24. vgef — *First Derivative Operator for Symmetric Exponential Filter (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Spatial Filters*
Operator:	*GEF Edge Extract*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

This Algorithm has been written by the team of:

Professor SERGE CASTAN

IRIT, URA 1399

118, route de Narbonne 31062 Toulouse FRANCE

OPTIMAL FILTER(ISEF) FOR EDGE DETECTION

PRINCIPLE :

I. Introduction

Edge detection is one of the most important subjects in image processing, which finds wide applications in the pattern recognition, the scene analysis and the 3-D vision, because the edges correspond in general to the important changes of physical or geometrical properties of objects in the scene and they are widely used as primitives in the pattern recognition, the image matching etc.

The edges coincide, generally speaking, grey level transition, they can be detected by maxima of gradient or the zero-crossing of the second derivatives calculated by some differential operators. Because the differential operators are sensitive to noise, a preprocessing such as smoothing is often necessary to eliminate the noise. A well-known smoothing filter is Gaussian filter and the edges can therefore be detected by a Laplacian-Gaussian filter. But there is an essential difficulty of the Laplacian-Gaussian filter which is the contradiction between the smoothing effect and the precision of edge localization. To overcome this difficulty, We proposed the optimal linear filter based on one step model (a step edge and the white noise) and the multi-edge model [9][10][11]. This optimal smoothing filter is a symmetric exponential filter of an infinitely large window size and can be realized by very simple recursive algorithm. It is proved that the band limited Laplacian of an input image filtered by this filter can be calculated from the Difference between the input and the output of this Recursive Filter (DRF). The edges detected by DRF method are less noisy and with a much better precision of localization.

The maxima of gradient or zeros of the second directional derivative along the gradient are a natural definition of intensity edges. Zeros of the Laplacian are only extensively used for their computational convenience. However, we must stress here that the zeros crossing of the Laplacian are not always coincided with the maxima of gradient, for example, the zeros of the Laplacian are farther apart than the maxima of gradient for circularly symmetric patterns, this lack of localization by the Laplacian can also be seen in the fact that zeros of Laplacian "swing wide" of corners. Therefore, it had better to detect the edges by maxima of gradient or zeros of the second directional derivative along the gradient .

So we propose two methods for edges detection, one uses maxima of Gradient (GEF), another uses the zeros crossing of Second directional Derivative along the gradient (SDEF).

II. The First and the Second Directional Derivative Operators for Symmetric Exponential Filter

A normalized symmetric exponential filter on 1-D can be written :

fL(x)=C*a0*(1-a0)**|x|=f1(x)#f2(x)=C*(f1(x)+f2(x)-a0*d(x)) (1)

where:

C=1/(2-a0), # means the convolution, d(x) is dirac function.

f1(x)={0 if x<0 and a0*(1-a0)**x if x>= 0 (2)

f2(x)={0 if x>0 and a0*(1-a0)**-x if x<= 0 (2)

we can write the first derivative operator of exponential filter:

fL'(x)=f2(x)-f1(x) (3)

And we can obtain the normalized second derivative operator of exponential filter :

fL"(x)=f1(x)+f2(x)-2*d(x) (4)

Because the exponential function is separable, we can write out 2-D exponential filter:

f(x,y) = fL(x)*fL(y) (5)

From the equations (18),(19) and (20), the first and the second directional derivative operators for symmetric exponential filter can be written like this:

fx(x,y) = fL(y)*(f2(x)-f1(x)) (6)

fy(x,y) = fL(x)*(f2(y)-f1(y)) (7)

fxx(x,y) = fL(y)*(f1(x)+f2(x)-2*d(x)) (8)

fyy(x,y) = fL(x)*(f1(y)+f2(y)-2*d(y)) (9)

III. The Recursive Algorithm for realizing the These Directional Derivative Operators of Symmetric Exponential Filter

The exponential filter is an IIR filter corresponding to an infinite window size, so we should realize the functions f1(x) and f2(x) (see formula (2)) by a recursive algorithm.

Supposing I(x,y) is the input image,
I1(x,y)=I(x,y)#f1(x) and I2(x,y)=I(x,y)#f2(x),
we have the recursive algorithm :

I1(x,y)=I1(x-1,y)+a0*(I(x,y)-I1(x-1,y)) (10)

I2(x,y)=I2(x+1,y)+a0*(I(x,y)-I2(x+1,y))

From the equations (1),(6),(7),(8) and (9), the band-limited first and second directional derivative of input image can be calculated by the recursive algorithm f1 and f2 as follows

Ix(x,y)=I(x,y)#f1(y)#f2(y)#(f2(x)-f1(x)) (11)

Ixx(x,y)=I(x,y)#f1(y)#f2(y)#(f2(x)+f1(x))-2*I(x,y)#f1(y)#f2(y) (12)

$$Iy(x,y) = I(x,y)\#f1(x)\#f2(x)\#(f2(y)-f1(y)) \quad (13)$$

$$Iyy(x,y) = I(x,y)\#f1(x)\#f2(x)\#(f2(y)+f1(y))-2*I(x,y)\#f1(x)\#f2(x) \quad (14)$$

With this algorithm, we can calculate at the same time the band-limited first and second directional derivative Ix and Ixx (or Iy and Iyy) of input image.

IV. Edges Detection

The band-limited first and second directional derivative of input image can be obtained by the algorithms as stated above. Using them, we can then realize the edge detection for an image.

The maxima of gradient or zeros of the second directional derivative along the gradient are a natural way of characterizing and localizing intensity edges, so we present here to detect the edges from the maxima of gradient or zeros of the second directional derivative along the gradient by using the differential operators of exponential filter.

1. Edges from the maxima of gradient

Using the first directional derivative operator of the exponential filter, the two band-limited first directional derivatives Ix and Iy can be calculated, and the gradient vector can be therefore determined approximatively for every point in image. The gradient magnitude image is then non maxima suppressed in the gradient direction and thresholded with hysteresis, i.e. if the entire segment of the contour lies above a low threshold T1, and at least one of part of which is above a high threshold T2, that contour is output. The non maxima suppression scheme requires three points, one of which will be the current point, and the other two should be estimated of the gradient magnitude at points displaced from the current point by vector normal to the edge direction.

2. Edges from the zero-crossings of the second directional derivative along the gradient direction

Because edges detected from local gradient maxima can not be a pixel width (less good precision of localization), we propose an other method which detect the edges from the zeros crossing of the second directional derivative along the gradient direction.

We can calculate Ix,Iy,Ixx and Iyy by using the method shown in paragraph III, and therefore obtain approximatively the gradient vector and the second derivative in the gradient direction for every point in image. We extract at first the zero crossing of second derivative along the gradient direction on which the gradient magnitude must be above a low threshold, so an edges image is obtained. To this image, the entire segment of the contour will be kept, if the gradient magnitude on at least one part of this contour is above a high threshold.

V. Comparison of Performance of the Filters

Filtering is a problem of estimation from noisy signal, and edge detection is a problem of estimating the position of maximal local signal change. Up to now, many works are done for edges detection in image, and different filters are proposed, for example, Gaussian filter, Canny filter, exponential filter, Deriche filter etc...

We appreciate the performance of the filters as follows :

(1) Precision of edge localization

According to our analysis [10], we can calculate the average localization error xe for Gaussian filter, Canny filter [15], Deriche filter [16] and the exponential filter :

xeG = (4*(2*e*3.14)**0.5)/a

xec = 0.81/a

xeD = 4*exp(-1)/a = 1.47/ a

xeE = 0

i.e. xeG > xeD > xec > xeE = 0.

So, we can see that the exponential filter localizes edge points with the best precision.

(2) Signal/Noise ratio on the edge point detected

Because xe is the average estimation for the position of the edge point detected, we propose to calculate Signal/Noise ratio (Eq.(7)) at the point xe.

And the signal/noise ratio for the Gaussian filter, Canny filter, Deriche filter and the exponential filter is :

SNRG = 2*s*exp(-32*s)/ ((3.14)**0.5)

SNRc = 0.39/ a

SNRD = 0.64/ a

SNRE =1/ a

i.e. SNRE > SNRD > SNRc > SNRG.

Then, we see that the exponential filter has the best noise eliminating effect among the above four.

(3) Complexity of calculation

For the complexity of calculation, we only tell the difference from exponential filter and Deriche filter [16], because they are implemented by recursive algorithms which have a simpler calculation.

Because ISEF can be realized by first order recursive filter, the ISEF algorithms are much simpler than that of Deriche filter. Besides, the ISEF algorithms can be implemented independently to every line and every column, it can be easily realized by a parallel system.

According to the analysis results above, the ISEF filter is superior to the others at the 3 principal aspects of the performance of the filter.

VI. Conclusion

The symmetric exponential filter of an infinite large window size is an optimal linear filter deduced from one step edge model and the multi-edge model, now we further prove that the symmetric exponential filter is the optimal edge detection filter in the criteria of the signal to noise ratio, the localization precision and unique maximum. Obviously, the real images will be still more complicated than these models, however DRF method has already provided good results for different type of images. The results obtained through the two new methods further show the superior performance of this filter. The theoretical analysis for the performance of the filters shows also that the exponential filter is superior to the other current filters.

The first and second directional derivative operators can be realized by recursive algorithm and calculated at the same time. The new algorithms are therefore very simple as well as DRF algorithm, and they are also easy to implemented in a parallel way.

Examples

vgef -i cross.xv -o output.xv

will compute a edge extraction by using the first directional derivative operator for symmetric exponential filter on image cross.xv and write the result in output.xv

For the GEF edge extraction the filter parameter value is generally a0 between 0.40 and 0.9 (0.45 [default] is a good value)

The hysteresis Thresholds T1 and T2 can be set to 12 and 15.

The minimum Length of segment is up to the user. It is set by default to 10.

Restrictions

works only with single band byte images.

Note that this routine was converted directly from Khoros 1.0. Due to the mixing of unsigned versus signed char, and the difference in the rules that resolve the mixing of these data types between ANSI C and the original K&R C that the code was written in, the results of running this program on an image will not be the same as with vdrf in Khoros 1.0. There is really no way to resolve which version is more correct, but the ANSI C version should at least be more consistent across machine architectures.

References

[1] W.K. PRATT, Digital Image Processing, New York, 1978.

[2] J. PREWITT, Object Enhancement and Extraction,Picture Processing and Psychopictories, Etd. by B. Lipkin and A. Rosenfeld, New York, pp.75-149, 1970.

[3] M. HUECKEL, An Operator Which Locates Edges in Digitized Pictures. J.A.C.M., Vol. 18, pp 113-125, 1971.

[4] R.O. DUDA and P.E. HART, Pattern Classification and Scene Analysis. Wiley, New York, 1973.

[5] R. HARALICK,Edge and Region Analysis for Digital Image Data. C.G.I.P., Vol. 12, pp 60-73, 1980.

[6] R.HARALICK and L.WATSON, A Facet Model for Image Data. C.G.I.P., Vol. 15, pp 113-129, 1981.

[8] D. MARR and E.C. HILDRETH, Theory of Edge Detection. Proc. R. Soc. Lond. B, Vol. 207, pp 187-217, 1980.

[9] J. SHEN and S. CASTAN, Un nouvel algorithme de detection de contours, proceedings of 5th Conf. on P.R.&.A.I. (in French), Grenoble, 1985.

[10] J. SHEN and S. CASTAN, An Optimal Linear Operator for Edge Detection. Proc. CVPR'86, Miami,1986.

[11] J. SHEN and S. CASTAN, Edge Detection Based on Multi-Edge Models.Proc. SPIE'87, Cannes, 1987.

[12] J. SHEN and S. CASTAN, Further Results on DRF Method for Edge Detection. 9th I.C.P.R., ROME, 1988.

[13] V.TORRE and T.A.POGGIO, On Edge Detection IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. Pami-8, N 2, March 1986.

[14] J.S.CHEN and G.MEDIONI, Detection, Localization, and Estimation of Edges. IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. 11, N 2, February 1989.

[15] J.F.CANNY, Finding Edges And Lines in Images. MIT Technical Report N 720, 1983.

[16] R. DERICHE, Optimal Edge Detection Using Recursive Filtering. In proc. First International Conference on Computer Vision, London, June 8-12 1987.

Command Line Arguments

```
Usage for vgef: First Derivative Operator for Symmetric Exponential Filter  (K1)
% vgef
  -i         (infile)  input image
  -o         (outfile) output image

  [-a0]      (float)   Exponential Filter Parameter for GEF
             (0 <= a0 <= 1, default = 0.45)
  [-t1]      (integer) 1st Hysteresis Threshold
             (0 <= t1 <= 255, default = 3)
  [-t2]      (integer) 2nd Hysteresis Threshold
             (0 <= t2 <= 255, default = 4)
  [-l]       (integer) Minimum Number of Pixels in a Segment
             (0 <= l <= 100000, default = 5)
```

B.25. vgettilt — *Compute the Best-Fit Plane Parameters for Input File (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Surface*
Operator:	*Find Tilt*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vgettilt computes the least-squares best fit plane for an image. The plane will be given by f(x,y)=Ax+By+C with the tilt coefficients in inverse pixels.

If the mask file is specified then only those points in the input image with a corresponding nonzero mask file pixel are used in the computation. The number of points actually contributing to the computation is given with the output data.

The output device or file is specified after the -f option, a filename of # will send the output to stderr.

Examples

vgettilt -i input -f tilt_info -m mask

See Also

vtilt, vdetilt

Restrictions

vgettilt will not operate on bit or complex data storage types, and will not work on multiband images.

Command Line Arguments

```
Usage for vgettilt: Compute the Best-Fit Plane Parameters for Input File  (K1)
% vgettilt
  -i         (infile)  input image
  -f         (outfile) output ASCII file

  [-m]       (infile)  statistics operation masking image
             (default = {none})
```

B.26. vgfractal — *Create Fractal Image with Specified Fractal Dimension (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Generate Data*
Operator:	*Fractal Image*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vgfractal creates a fractal image with a desired fractal dimension. A fractal image is created using the midpoint displacement algorithm, which uses a recursive generation technique. Numerous types of fractal images may be created by varying the input parameters.

The output image, which corresponds to the -o argument, is stored as data type FLOAT, and will be a single band image. The size of the resulting image will be determined by the maximum number of recursions. That is, the resulting image will be square, with dimensions (2ˆrecursions +1) by (2ˆrecursions +1).

The input parameters include: maximum number of recursions, desired hurst coefficient, standard deviation of random numbers, seed, coastlines, and random additions. Descriptions on the use of these parameters follow.

The maximum number of recursions (iterations) is specified by the -r recursive iteration argument. The recursive iteration argument must be an integer ranging from 2 to 10. The size of the image will be a power of two of the recursive iteration argument. That is, it will create an image with sizes ranging from (5 by 5) to (1025 by 1025). The default number of recursions is 6, which will result in an image of size 65 by 65 pixels.

The hurst coefficient of the generated image may be altered with the -h argument. This allows the user to change the fractal dimension of the generated image. Since the hurst coefficient is a measure of the slope of window size vs range in pixel values, the valid range for the hurst coefficient is $0 < H < 1$.

The fractal dimension is calculated from the hurst coefficient as follows:

$$D = 3 - H$$

where,

D is the fractal dimension

H is the hurst coefficient.

The parameter 3 reflects the 3-dimensional Euclidean geometry of the system (ie. X, Y, and Z or pixel value).

The standard deviation of the randomly generated numbers used to create the fractal image may be altered using the -s argument. Increasing the standard deviation will permit a wider range of values to be generated by the random number generator. This will have the same effect as scaling the values that are generated.

A seed may be entered to allow the same image to be generated, or the system time may be used to set the seed (default).

The coastline argument (-c) can be toggled on (default is off), which will effectively generate a "coastline" by setting all generated values that fall below zero to be equal to zero. This creates the effect of a body of water running up against a landform, or a "coastline".

The random additions argument (-m) can be toggled on, which will allow random elements to be added to each point, whereas with random additions turned off (default), random elements are added only to the new midpoints generated in each stage.

Note that in order to display the image as a three dimensional relief plot, the data can be plotted in xprism.

Theory - The midpoint displacement algorithm constructs the image using a recursive generation technique. This method works with square lattices of points or pixels. Given a resolution (w) of such a lattice or grid, another square grid is obtained with a resolution of w/sqrt(2) by adding the midpoints of all squares. The orientation of the new lattice is rotated by 45 degrees, and adding the midpoints of all squares yields the next lattice with a resolution of w/2 and the same orientation as the first. This is illustrated as follows:

```
o   x   o           o   o   o          o x o x o
                      x   x            x o x o x
x   o   x   ---->   o   o   o   ---->  o x o x o
                      x   x            x o x o x
o   x   o           o   o   o          o x o x o

where,      o = old points
            x = new points
```

The trigger input can be used to cause *vgfractal* to re-execute when used inside cantata. No image is read from this input.

Examples

vgfractal -o output_image.xv -r 5 -h 0.7 -d 2.0 -s 1359 -m 1

the selected number of recursions is 5 (2^5 +1 or a 33 x 33 image). The hurst coefficient selected is 0.7, which results in an image with a fractal dimension of 2.3. The standard deviation is 2.0 (slightly wider range of values generated). The seed for the random number generator is entered as 1359. The random additions flag is toggled on.

See Also

vfractal(1)

Restrictions

The resulting output image will be of type FLOAT.

References

vgfractal uses the Midpoint Displacement Algorithm; Benoit Mandelbrot discusses it in "The Fractal Geometry of Nature", W.H. Freeman, 1983, pp. 43b, 233, and 263.

For implementation details, see H.-O. Peitgen and D. Saupe, The Science of Fractal Images, Springer-Verlag, 1988.

Command Line Arguments

```
Usage for vgfractal: Create Fractal Image with Specified Fractal Dimension  (K1)
% vgfractal
  -o         (outfile) Output image filename

  [-trigger] (infile)  trigger input
             (default = {none})
  [-r]       (integer) Number of recursions
             (2 <= r <= 10, default = 6)
  [-h]       (float)   Hurst coefficient
             (0 <= h <= 1, default = 0.5)
  [-d]       (float)   Standard deviation
```

```
               (d > 0.0, default = 1)
  [-s]         (integer) Seed
               (s >= 0, default = 0)
  [-c]         (boolean) Generate coastline
               (default = false)
  [-m]         (boolean) Add random elements to each point
               (default = false)
```

B.27. vgpwl — *Create 2D Piecewise Linear Periodic Function (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Generate Data*
Operator:	*Piecewise Linear*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

creates a two dimensional piecewise linear image. The image pixel values follow a square waveform. The user can specify the period, phase, minima and maxima, sampling frequency, rise and fall times, and image dimensions.

NOTE: According to the Sampling Theorem, the sampling frequency should be at least twice that of the signal frequency to generate a signal properly.
Note that the X-direction corresponds to across the image from left to right, and the Y-direction corresponds to down the image from top to bottom.

The trigger input can be used to cause *vgpwl* to re-execute when used inside cantata. No image is read from this input.

Examples

vgpwl -o plinear.xv -t float
vgpwl -o test.xv -t complex -xr 10 -yr 10 -xw 5 -yw 7

Restrictions

can only generate VFF_TYP_FLOAT and VFF_TYP_COMPLEX type images.

Command Line Arguments

```
Usage for vgpwl: Create 2D Piecewise Linear Periodic Function  (K1)
% vgpwl
  -o           (outfile) resulting image

  [-trigger]   (infile)  trigger input
               (default = {none})
  [-r]         (integer) number of rows
               (r > 0, default = 512)
  [-s]         (float)   sampling frequency
               (s > 0.0, default = 2)
  [-c]         (integer) number of columns
```

```
              (c > 0, default = 512)
  [-l]        (float)   signal minimum
              (unbounded, default = 0)
  [-b]        (integer) number of bands
              (b > 0, default = 1)
  [-m]        (float)   signal maximum
              (unbounded, default = 1)
  [-xp]       (float)   signal period in X-direction
              (xp > 0.0, default = 1)
  [-yp]       (float)   signal period in Y-direction
              (yp > 0.0, default = 1)
  [-xr]       (float)   rise time in X-direction
              (xr >= 0.0, default = 0)
  [-yr]       (float)   rise time in Y-direction
              (yr >= 0.0, default = 0)
  [-xf]       (float)   fall time in X-direction
              (xf >= 0.0, default = 0)
  [-yf]       (float)   fall time in Y-direction
              (yf >= 0.0, default = 0)
  [-xw]       (float)   pulse width in X-direction
              (xw >= 0.0, default = 0)
  [-yw]       (float)   pulse width in Y-direction
              (yw >= 0.0, default = 0)
  [-t]        (string)  lets user choose data type of image
           float  (select data type),
           or complex  (choose data type)
[default: float]
```

B.28. vgrow — *Compute the Median Axis Inverse Transform (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Segmentation*
Operator:	*Inverse Median Axis*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vgrow computes the median axis inverse transform.

This transformation allows the recovery of the original image from a median axis image.
This routine is useful for displaying the modifications introduced after any transformation of the median axis.

This transformation assigns the value 255 to any pixel whose distance to at least one median axis pixel, is less than or equal to the value of this pixel.
The user is referred to the man pages of the median axis routine as an aid in understanding the inverse transform.

ADVICE:

For this routine, the image must be a byte image, with the gray level value of 0 for the background. The pixels that do not belong to the background are supposed to belong to a median axis, so it is advisable to use an image obtained from the median axis transform.

Examples

vgrow -i image1 -o image2
Computes the median axis transform.
For this routine, the image must be a median axis representation of an image (obtained from vmediaxis).

See Also

lvgrow(3), lvdistance(3), vdistance(1), lvmediaxis(3), vmediaxis(1)

Restrictions

vgrow works only on BYTE images that represent a coherent median axis of an image.

Command Line Arguments

```
Usage for vgrow: Compute the Median Axis Inverse Transform  (K1)
% vgrow
  -i         (infile)  input image
  -o         (outfile) output image
```

B.29. vhmed — *Median Filter the Image Using Histogram to Find Median (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Spatial Filters*
Operator:	*Median Histogram*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vhmed computes a two dimensional median filter of width w and height h over the given image. The median is computed by moving a w by h pixel mask over the image and updating the histogram of the mask.

Images must be of data storage type byte.

In rectangular odd windows (masks) the median value is replaced in the center pixel location of the window. The default window size is 3x3.

The width or height of a window cannot be an even number.

Examples

vhmed -i input.image -o output.image -w 3 -h 3

This command performs median filtering of width 3 and height 3 on input.image and stores the result in output.image.

See Also

vqmed(1)

Restrictions

vhmed does not support even window dimensions and only works on data storage type byte.

Command Line Arguments

```
Usage for vhmed: Median Filter the Image Using Histogram to Find Median  (K1)
% vhmed
  -i         (infile)  input image filename
  -o         (outfile) output image filename

  [-w]       (integer) window width argument
             (1 <= w <= 32, default = 3)
  [-h]       (integer) window height argument
             (1 <= h <= 32, default = 3)
```

B.30. vhxray — *Stretch or Equalize Histogram Using Overlapping Windows (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Histogram Operators*
Operator:	*Window Eq/Str*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vhxray will enhance an image by using a histogram stretch or histogram equalization method. Vhxray will use overlapping windows over the entire image, and for each window compute the histogram and perform a stretch or equalization. The middle pixel of the window is updated according to the new histogram value that corresponds to the old image pixel values.

The window size must be odd (3x3, 5x5, 15x15, etc), since we must have a center pixel to update for each window. The default window size is 5x5.

The input image must be of type BYTE. The default operation is to use the histogram equalization method (-t = 1).

Since windows are being used, the border pixels will not be updated. The border pixels will be set to zero, black. The border size can be calculated as follows: window width / 2 = side border, window height /2 = top & bottom border.

A large window size will give better results than a small window size. For example a window size of 33 x 33 on a 512 x 512 image will give better results than a window size of 5 x 5 on the same image.

Examples

vhxray -i input.image -o output.image -w 9 -h 9 -t 0

This command performs image enhancement on input image using histogram equalization and stores the result in output.image. The size of the window used is 9 by 9.

See Also

khistops, khstr, kheq

Restrictions

vhxray Will only operate on BYTE images. The image border is not updated as explained above.

Command Line Arguments

```
Usage for vhxray: Stretch or Equalize Histogram Using Overlapping Windows  (K1)
% vhxray
  -i         (infile)  first input image
  -o         (outfile) resulting output image

  [-w]       (integer) selects the width of the window
             (1 <= w <= 32, default = 5)
  [-h]       (integer) selects the height of the window
             (1 <= h <= 32, default = 5)
  [-t]       (boolean) Choose Histogram Stretch (true) or Equalization (false)
             (default = false)
```

B.31. vinverse — *Image Restoration by Inverse Filtering in Frequency Domain (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Frequency Filters*
Operator:	*Inverse Filter*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

NOTE: Program will NOT execute! vinverse functionality is replaced by DATAMANIP::kinverse. Command line arguments have changed slightly. The -o filtered data object (FFT) output option is now -o1. To reproduce the vinverse default settings, run kinverse with the following arguments.

%kinverse -t 1.0 . . .

vinverse performs restoration of images in the spatial domain by inverse filtering in the frequency domain. The idea is to deconvolve the blurring function from the blurred image. A few problems exist in performing this operation, as described in great detail in any good image processing book.

The major difficulty lies in how small the denominator is allowed to be before you call it garbage. The value chosen is supplied as the -t thresh value. Ideally, this will be just above the noise floor in the image, but experimentation is the best guide here.

The reffile (reference image) is an image which, if supplied, is subtracted from the input image before the inverse filter is applied. This is handy for removing some constant background from the image before the restoration is done.

The PSFfile (PSF function) is the transform of the Point-Spread Function, or blurring function that is to be deconvolved from the input image.

The effPSFfile (effective PSF function) is the same as the PSFfile after the thresholding operation has been applied. Taking the inverse Fourier transform of this file gives the *effective* PSF as used in the restoration. The idea is that is should be reasonably close to the real PSF.

Examples

vinverse -i1 fft_img1 -i2 fft_psf_img -t 10 -o1 fft_out_img

This command deconvolves the second input image from the first. Fft_img1 is the FFT of the image to deconvolve from, and fft_psf_img is the FFT of the PSF function of the image to deconvolve. The threshold for the denominator is 10, and the resulting FFT image is fft_out_img.

See Also

DATAMANIP::kinverse

Restrictions

all images must be of the same data type (VFF_TYP_COMPLEX) and size.

Command Line Arguments

```
Usage for vinverse: Image Restoration by Inverse Filtering in Frequency Domain  (K1)
% vinverse
  -i1        (infile)  FFT of image to be restored
  -i2        (infile)  FFT of PSF function
  -o1        (outfile) FFT of restored image

  [-i3]      (infile)  FFT of reference image
             (default = {none})
  [-t]       (float)   minimum PSF threshold
             (t > 0.0, default = 0.1)
  [-o2]      (outfile) FFT of effective PSF function
             (default = {none})
```

B.32. vkmeans — *Perform the K-means Clustering Algorithm on Image (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Classification*
Operator:	*K-Means*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

NOTE: Program will NOT execute! vkmeans functionality is replaced by the DATAMANIP::kkmeans operator. The following command line arguments have changed.

-f2, Output file for kmeans statistics, is now -o5
-b, Specifies the border width in pixels deleted
-m, Create map in cluster number output image, is now -map
-f1 (input ASCII file for x-y coordinates of cluster centers) and -c (use First K Vectors As Cluster Centers) deleted. If -i2 is not specified, default to -c behavior

vkmeans converts an input image into vectors of equal size and performs the kmeans clustering algorithm on the vectors using the K initial cluster centers. The size of each vector is determined by the number of bands in the input image.

The K-means algorithm is based on the minimization of the sum of the squared distances from all points in a cluster to a cluster center. The user chooses K initial cluster centers and the image vectors are iteratively distributed among the K cluster domains. New cluster centers are computed from these results, such that the sum of the squared distances from all points in a cluster to the new cluster center is minimized.

Although the k-means algorithm does not really converge, a practical upper limit is chosen for convergence. The user has the option of specifying the maximum number of iterations using the -n option. The default is 100000 iterations.

There are three options available for entering initial cluster centers. These are mutually exclusive and optional:

-c uses the first K vectors as initial cluster centers (default).

-f1 uses an ASCII file as input for the x-y coordinates of the initial cluster centers. The ASCII file must contain an xy pair specifying the x-y coordinate for each initial cluster center. The number of xy pairs MUST equal the value of the -k option.

-i2 uses an input image as the initial cluster center values. This image MUST contain the same number of data bands as the input image. The number of cluster centers in the image MUST equal the value of the -k option.

The -o2 optional argument will assign the cluster center image to a file. The cluster center image will always have a row size of 1 and a column size of the number of clusters, the number of data bands will be equal to the dimension of the vectors.

The -o3 optional argument will assign the variance image of the cluster centers to a file. The variance image will follow the same format as the cluster center images.

The Statistics file (-f2) contains statistics obtained during the execution of vkmeans. This file includes the following information:

Total Number of K-means Iterations
Total Number of Clusters
Number of Vectors Per Cluster
Cluster Center Values
Cluster Center Variance Values
Trace of Covariance Matrix

If the -m (map flag) optional argument is set, the cluster number image will contain a map of type FLOAT. The map contains the cluster center values. The number of rows in the map is equal to the number of desired clusters, while the number of columns is equal to the dimensionality of the data. Also, when the -m argument is set, the ispare1 field is set to 0 and the ispare2 is set to the dimension of the vector. By setting the -m argument to true, the output of vkmeans is compatible with the spectrum classification application.

The border option (-b) allows the user to specify a border width, in pixels, encompassing the desired region of the image to be clustered using the k-means algorithm. The border region will contain the value zero. This region is ignored during the clustering process, and the cluster center values do not reflect this area.

Results obtained by the k-means algorithm can be influenced by the number and choice of initial cluster centers and the geometrical properties of the data.

All input images must be of data storage type FLOAT.

This routine was written with the help of and ideas from Dr. Don Hush, University of New Mexico, Dept. of EECE.

Examples

vkmeans -i1 image1 -c -n 10000 -k 6 -b 4 -o1 image2 -o2 image3

this will apply the k-means clustering algorithm to image1 using the first K vectors as cluster centers. The number of iterations selected is 10000, and the number of clusters selected is 6, and a border width of 4 pixels is specified. Image2 is specifies which vector belongs to what cluster, while image3 is the output image for the cluster centers.

vkmeans -i1 image1 -f1 file1 -k 8 -o1 image2 -o2 image3 -f2 file2 this will apply the k-means clustering algorithm to image1 using the x-y coordinates of cluster centers specified in file1. The -k option specifies 8 cluster centers, which MUST correspond with the number of cluster centers in file1. An ASCII file containing the k-means statistics (file2) is created.

vkmeans -i1 image1 -i2 image2 -k 8 -o1 image3 -o2 image4

this will apply the k-means clustering algorithm to image1 using an input image2 specifying the cluster centers. The -k option specifies 8 cluster centers, which MUST correspond with the number of cluster centers in image2.

See Also

DATAMANIP::kkmeans

Restrictions

All input images must be of data storage type FLOAT.

References

None

Command Line Arguments

```
Usage for vkmeans: Perform the K-means Clustering Algorithm on Image  (K1)
% vkmeans
  -i1        (infile)  input image
  -o1        (outfile) Output cluster number image
  -f2        (outfile) Output file for kmeans statistics

  [-n]       (integer) Specifies maximum number of iterations
             (1 <= n <= 100000, default = 50000)
  [-k]       (integer) Specifies number of clusters
             (k > 0, default = 2)
  [-b]       (integer) Specifies the border width in pixels
             (b >= 0, default = 0)
  [-m]       (boolean) Create map in cluster number output image
             (default = true)
  [-o2]      (outfile) Output image for cluster centers
             (default = {none})
  [-o3]      (outfile) Output image for variance
             (default = {none})

  [Optional Mutually Exclusive Group - Specify one or none of the 3 options:]
    [-c]  (boolean) if Yes use First K Vectors As Cluster Centers
          (default = true)
    [-f1] (infile)  input ASCII file for x-y coordinates of cluster centers
          (default = {none})
    [-i2] (infile)  input image cluster center values
          (default = {none})
```

B.33. vlabel — *Perform Labeling on Multiband or Cluster Image (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Classification*
Operator:	*Labeling*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vlabel performs a labeling on a multiband image or a cluster image by attempting to merge connected pixels.

The principal of the algorithm is as follows: A pixel receives the same label as its neighbor if the likelihood distance between the two pixels is acceptable. The label process is propagated for a given region number until it is no longer possible to find a candidate.

Three different types of labeling choices exist:

1 *First*choice: uses a single or multi band image, where the data storage can be any of the following types: VFF_TYP_1_BYTE, VFF_TYP_2_BYTE, VFF_TYP_4_BYTE, or VFF_TYP_FLOAT.

The input image corresponds to the -i1 argument.

The distance is computed using all the bands of the image.

2. *Second*choice: uses a cluster number and cluster center image obtained from an algorithm like vkmeans or vquant. For this case, the cluster center represents the value of a class of pixels that have been grouped together. Therefore, the distance between two neighbors will be the distance between their clusters.

This case will require less computation time because the algorithm will only compute the interclass distance, instead of computing the distance of two neighbors for the entire image.

An additional advantage with this choice, is that the output from algorithms such as vkmeans or vquant may be utilized, which may lead to better results.

The cluster center image corresponds to -i2 argument
The cluster number image corresponds to -i3 argument in the command line.

3. possibility*The*final is to use a single or multiband input image (argument -i1) associated with a cluster number image (argument -i3).

The advantage of this choice is that the results of a clustering algorithm are used to keep the neighbor pixels that have the same cluster number in the same class, and to rely on the distance in the single or multi band image to group two neighbors that do not belong to the same cluster.

In summary, the three possible choices are:

-i1 image.xv

-i2 cluster_center.xv -i3 cluster_number.xv

-i1 image.xv -i3 cluster_number.xv

also*The*algorithm the following parameters:

Metric distance: There are 2 different metric distances that can be used.

-d 1 uses Euclidean distance: sqrt[(x-s)^2 + (y-t)^2].

-d 2 uses City Block distance: |x-s| + |y-t|.

Connectivity: There are two possible neighborhoods:

-c 1 uses the 4 connectivity to link pixels together.

-c 2 uses the 8 connectivity.

Minimum size of a region:

-s (float_value) determines the number of pixels required for a region to be retained. The minimum number of pixels is equal to:
Total number of pixels in image * float_value / 100.0 (-s corresponds to a percentage of the total number of pixels in the image).

Border Size:
Each pixel in the image is updated except those outside of the border. The size of the border is specified by the -w argument.

Merging Process:
When the labeling process is computed, the user can expect that the small rejected region will be merged together in a bigger acceptable region or will be included inside another connected region.
This choice is selected by setting the logical argument -merge to TRUE. If -merge is set to FALSE, the

small regions will be ignored and labeled as an UNDEFINED REGION (label number 0), the same as the border.

The AUTOMATIC or MANUAL OPTION:
This option allows the user either to fix a threshold, or to give an approximate number of regions.
If the AUTOMATIC option is used, the algorithm will iterate on the threshold until the number of regions labeled by the process is comparable to the number of expected regions.
In fact, if the expected number is not reached after 30 iterations, the threshold that gives the closest number of regions is used for the final labeling.

Although this option is easy to use, the function:

number of regions = F(Threshold) is not a monotonically increasing function, and the convergence toward a solution may not exist.

-n int_value (AUTOMATIC OPTION) determines the approximate final number of regions expected.
-f float_value (MANUAL OPTION) determines the threshold used by the labeling process. THESE TWO OPTIONS ARE MUTUALLY EXCLUSIVE

The default threshold value is 0.07. This value generally gives good results on noiseless images with large uniform regions.
Decreasing this value will increase the number of regions found during the labeling process, but these regions will get smaller and could be rejected by the minimum size threshold.
Increasing this value will decrease the number of regions found during the labeling process. At the same time, the number of small regions will decrease which means that this area of the curve, **number of regions = F(Threshold)**, is more stable than the other one. Once the user becomes accustomed to this routine, good results are generally obtained. One way to become familiar with the routine, is to use the automatic option and analyze the output ASCII file (Statistics on the iteration process). This file contains the number of regions labeled for each iteration, allowing the user to see how the number of regions changes as the THRESHOLD is changed.

Output Files

1 **Output image:** The resulting image, which corresponds to the -o1 argument, contains the labeled image in which every pixel has a region number as its value. This image is of data storage type VFF_TYP_4_BYTE.
The region label numbers are 1 to N. The region number 0 is reserved as an UNDEFINED label or for the border.

2 **Output Statistic ASCII File:** This file contains all the information relative to the labeling process.

Examples

vlabel -i1 image.xv -d 1 c 1 -merge 0 -f 0.07 -s 0.7 -w 2 -o image1.xv -asc stats This command will label image.xv using the Euclidean distance, the 4 connectivity, a split and merge factor equal to 0.07, a minimum size for the regions equal to 0.7 percent of the total number of pixels in image.xv, and a border size of 2. The merge option is not used which means that the small regions will not be labeled. The labeled image will be stored in image1.xv and the statistics will be written in the ASCII file, stats.

See Also

lvkmeans(3), lvquant(3), vkmeans(1), vquant(1).

Restrictions

vlabel works only with cluster number and cluster center images conforming to the convention established by the set of clustering algorithms.

Command Line Arguments

```
Usage for vlabel: Perform Labeling on Multiband or Cluster Image  (K1)
% vlabel
  -o         (outfile) output image after the labeling process

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i1] (infile) input image
          (default = {none})

    [Optional Mutually Inclusive Group - Specify all or none of the 2 options:]
      [-i2] (infile) input image for cluster centers
            (default = {none})
      [-i3] (infile) input image for cluster numbers
            (default = {none})

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-n] (integer) Number of region to get from labeling process
         (2 <= n <= 2500, default = 10)
    [-f] (float)   Split & Merge Factor: higher values result in finer regions
         (0 <= f <= 1, default = 0.07)

  [-w]       (integer) Specify the border width in pixels
             (0 <= w <= 16, default = 0)
  [-m]       (integer) Distance Metric: 1 = Euclidean distance, 2 = city block
          1  (Use Euclidean Distance),
          or 2  (Use City block Distance)
[default: 1]
  [-c]       (integer) Connectivity: 1 = 4-connectivity, 2 = 8-connectivity
          1  (use 4 Connectivity),
          or 2  (use 8 Connectivity)
[default: 1]
  [-s]       (float)   Minimum number of pixel to keep in label region
             (0 <= s <= 100, default = 1)
  [-merge]   (boolean) if flag= 1 the small regions will be merged together in connected regions, or will be included to existing labeled regions
```

```
              (default = false)
    [-asc]    (outfile) Output ASCII file for vlabel result display
              (default = {none})
```

B.34. vmarr — *Create a Marr Type Edge Detection Filter Kernel (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Generate Data*
Operator:	*Marr Filter*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

creates a convolution filter kernel using the following equation. pixel = [-1/(PI*var*var)]*[1-r*r/(2*var)]*exp[-r*r/(2*var)], where r is the calculated distance from the center of the kernel.

A Marr filter is the Laplacian of a Gaussian, and is intended for high lighting image edges and at the same time smooth out the noise. vmarr creates image data of type float.

See Also

kasc2val, kimportasc

Restrictions

only generates float kernels, kconvert can change the image to other data types.

Command Line Arguments

```
Usage for vmarr: Create a Marr Type Edge Detection Filter Kernel  (K1)
% vmarr
  -o         (outfile) Marr kernel output image

  [-trigger] (infile)  trigger input
             (default = {none})
  [-r]       (integer) specify the number of rows in the kernel
             (r > 0, default = 3)
  [-c]       (integer) specify the number of columns in the kernel
             (c > 0, default = 3)
  [-v]       (float)   variance of the Gaussian
             (v > 0.0, default = 1)
```

B.35. vmediaxis — *Compute the Medial Axis Transform (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Segmentation*

Operator: *Medial Axis Transform* Object Type: *kroutine*
In Cantata: *Yes*

Description

vmediaxis Computes the medial axis transform on a byte binary image.

The median axis of a region in a binary image is the set of pixels whose minimum distance from the edge is a local maximum. It is a thinning algorithm that is comparable to the morphological skeletonization. The median axis is generally not connected.

An optional flag allows the user to obtain the medial axis pixel values set either to their distance from the border, or to 255. If the flag is set to one, it leads to a better visualization of the median axis but the information that permits the user to reconstruct the original binary image is lost.
If the flag is not set to 1, it is possible to reconstruct the original image from the medial axis.

Presentation of the **medial axis transform** algorithm :

U(m,n) : *original image*. This image must be a two-valued byte image, 0 for the background, and any other value for the objects.

Uk(i,j) : *image after k iterations*.

U0(m,n)= U(m,n) , and for k= 1, 2, 3, ...
Uk(m,n)= U(m,n)+min{U0(i,j);((i,j):dist(m,n;i,j)<= 1)},
dist(m,n;i,j) is the mediaxis between (m,n) and (i,j).
dist(i,j-1;i,j)=dist(i,j+1;i,j)=dist(i-1,j;i,j)=dist(i+1,j;i,j)=1.
dist(i-1,j-1;i,j)=dist(i+1,j-1;i,j)=dist(i-1,j+1;i,j)=dist(i+1,j+1;i,j)=2.

The transform is done when k equals the maximum thickness of the region, that is when Uk+1 equals Uk.

The last part consists of using this image and keeping only the pixels that are a local maximum in the 4 neighborhood.
The **Medial Axis image** is a form of **skeletonization**.

Functions that can be useful when using vmediaxis:

threshold
generates a binary image.

ADVICE:

For this routine, the image must be a byte binary image, with the gray level value 0 for the background. So it is advisable to threshold the image before using vmediaxis.

Examples

vmediaxis -i image1 [-w wflag] -o image2
Computes the medial axis transform.
For this routine, the image must be a byte binary image, with the gray-level value 0 for the background.
So it is advisable to threshold the image before using vmediaxis.

See Also

lvmediaxis(3), kclip

Restrictions

vmediaxis works only with binary BYTE images.

Command Line Arguments

```
Usage for vmediaxis: Compute the Medial Axis Transform  (K1)
% vmediaxis
  -i        (infile)  input image
  -o        (outfile) output image

  [-w]      (boolean) if flag=1 the medial axis has the value 255, otherwise (default) the value is the distance from the border
            (default = false)
```

B.36. vmindis — *Minimum Distance Classifier (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Classification*
Operator:	*Minimum Distance*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vmindis is a simple minimum distance classifier. The distance metric used to the Euclidean distance. This routine accepts two images as the input. The image that corresponds to the -i1 argument is the image that needs to be classified. This image can be have as many data bands as necessary. The number of data bands depicts the dimensionality of the vectors. Each pixel in the image is a vector with dimensionality of the number of data bands.

The other input image which corresponds to the -i2 argument is the prototype image. The last data band of this image is the class data band. This image must contain the same number of data bands as the other input image plus an extra data band that represents the class mapping. This image would most likely have been created by vkmeans or some other routine that will give cluster centers. This image contains vectors that correspond to the prototype of each class.

As stated above the center image's last data band is a class data band. The class data band simply maps each vector in the center image to the final class.

At this point most class images must be created manually. A class image can be created by using pseudo color in editimage on the resulting clustered image from the vkmeans routine. Vcustom can then be used to create the class image. Finally kinset can be used to add the class image data to the center image. Normally one would over cluster using vkmeans then reduce the space down. So vkmeans might produce 100 clusters but, really only 3 clusters are desired. Pseudo color in editimage and kasc2val can be used to reduce the space down and create a class image. The class data might map cluster center vectors 1-50 to class 1, vectors 51-60 to class 2 and vectors 61-100 to class 3. When the vmindis routine is run, the result would be a classified image of three classes.

The border option (-b) allows the user to specify a border width, in pixels, encompassing the image. The border region is skipped by vmindis when classification is performed. This useful if neighborhood operators have been used previously and have not updated edge pixels.

All input images must be of data storage type FLOAT.

Examples

vmindis -i1 aerial_image.xv -i2 vkmeans_centers -b 5 -o test

aerial_image.xv is a 5 band image; thus, each vector has dimensionality of 5, vkmeans_centers image is a 6 band image, 5 bands are the cluster center values produced by vkmeans and 1 class band. The border that vmindis will ignore is 5 pixels wide. the output image will be a single band image called test.

Restrictions

All input images must be of data storage type FLOAT.

Command Line Arguments

```
Usage for vmindis: Minimum Distance Classifier  (K1)
% vmindis
  -i1        (infile)  input image to be classified
  -i2        (infile)  input center/class image
  -o         (outfile) output specifying which vector belongs to which cluster

  [-b]       (integer)  specifies the border width in pixels (default = 0)
             (0 <= b <= 100, default = 0)
```

B.37. vmos2band — *Convert Mosaic Image to Multiband Image (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Organize Data*
Operator:	*Mosaic to Multiband*	Object Type:	*kroutine*

In Cantata: *Yes*

Description

vmos2band converts a mosaic input image to an output image in which each subimage of the input mosaic constitutes a band of the output image.

The output image generated by vmos2band will be (r-2*bs) rows by (c-2*bs) columns with n bands, where r and c are the number of rows and columns per subimage, bs is the subimage border size, and n is the number of subimages that are to be put into band format. Vmos2band assumes that all subimages are same size (r*c) and all subimages have the same border size (bs). Subimages from the input mosaic image are assigned to bands in the multiband output image in the following manner: The first subimage specified is assigned to the first band in the output image. Subsequent bands are assigned subsequent subimages indexed from left to right, top to bottom. Vmos2band accepts VIFF data_types (excluding VFF_TYP_BIT) and returns that same data_type. If a map is associated with the input mosaic image, the same map will be assigned to the output multiband image.

Vmos2band does not work on explicit location data and will return an error when such an image is encountered. No output image will be created.

number of columns and number of rows in subimages
The -r and -c options are used to specify the number of rows and the number of columns, respectively, per subimage. Vmos2band assumes that all subimages in the mosaic image are the same size. The -r and -c values SHOULD include any borders associated with the subimages. The default subimage row and column sizes are 16 pixels.

subimage border size
The -bs option allows the user to specify the subimage border size in pixels. Again, this parameter will be applied to all subimages in the mosaic. When vmos2band converts the mosaic image to a multiband image, the subimage borders will be removed from the subimages before storing them in bands. The size of the output image will therefore be (r-2*bs) rows by (c-2*bs) columns by n bands. The default subimage border size is 0 pixels.

overall mosaic image border size
The -bm option allows the user to specify, in pixels, the size of an outer border that should be removed from the mosaic image before extracting the subimages. The overall mosaic image border and the subimage borders are independent. The default overall mosaic image border size is 0 pixels.

horizontal and vertical offset of first subimage to be banded
The user can specify the first subimage in the mosaic that will be put into the multiband output image by using the -x (horizontal offset) and -y (vertical offset) options. The specified subimage will then be the first band in the output image. The indices, x and y, index through the subimages (they are NOT indices through the input mosaic image rows and columns). Therefore, if a mosaic image contained 9 4-row by 4-col subimages arranged as 3 subimages in the horizontal direction by 3 subimages in the vertical direction, the center subimage would be indexed by x=1, y=1 (not by x=4, y=4). The default first subimage is the first subimage in the mosaic (x=0, y=0).

number of subimages to be put into band format
The user can also specify how many images will be put into band format, starting with the first subimage (described above), by specifying the -n option. If the specified number of subimages cannot be extracted, vmos2band will return an error and the output image will not be created. If the -n option is

not specified, vmos2band calculates the number of subimages that will be banded, taking into account both the subimage skip value and the first subimage offsets.

subimage skip value
The subimage skip value option (-s) allows the user to specify that the next s subimages be skipped each time a subimage is stored as a band. For example, a skip value of s=1 in a 3x3 subimage mosaic would band the subimages in a checkerboard pattern (every other subimage is a band in the output image). When applying the subimage skip value, vmos2band looks at the series of subimages as a linear array. Therefore, the row/column arrangement of subimages does not affect how the skipping occurs.

Examples

vmos2band -i mosaic.xv -o band.xv -r 18 -c 18 -bs 1 -n 3
Will return a 3 band image of dimension 16 rows by 16 cols. Since there is a border of 1 (-bs 1) around the image, the resulting image is 16 rows by 16 cols.

Restrictions

Vmos2band does not work on VFF_TYP_BIT data files.

Vmos2band does not work on explicit location data and will return an error when such an image is encountered. No output image will be created.

Command Line Arguments

```
Usage for vmos2band: Convert Mosaic Image to Multiband Image  (K1)
% vmos2band
  -i         (infile)  input image
  -o         (outfile) resulting output image

  [-r]       (integer) number of subimage rows (must be equal for all subimages)
             (r >= 0, default = 16)
  [-c]       (integer) number of subimage columns (must be equal for all subimages)
             (c >= 0, default = 16)
  [-bs]      (integer) Subimage border size in pixels (inner borders)
             (bs >= 0, default = 0)
  [-bm]      (integer) Mosaic image border size in pixels (outer border)
             (bm >= 0, default = 0)
  [-x]       (integer) Horizontal index of first subimage (will be first band)
             (x >= 0, default = 0)
  [-y]       (integer) Vertical index of first subimage (will be first band)
             (y >= 0, default = 0)
  [-s]       (integer) Number of subimages to skip (periodic) when putting subimages into band format
             (s >= 0, default = 0)
  [-n]       (integer) Number of subimages to be put into band format
             (n >= 0, default = 1)
```

B.38. vpml — *Estimate Fractal Dimension of Image Based on P(m,L) (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Feature Extraction*
Operator:	*Fractal Dim P(m,L)*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vpml estimates the fractal dimension of an image based on the probability that there are m pixels within a window of size L centered on a pixel from a particular class. For a selected range of window sizes (L), the window is centered on the first occurrence of the pixel belonging to a particular class. The number of pixels within a window of size L, belonging to a specific class are counted (including the center pixel of the window), and a "histogram" is formed as the window is moved over the image. This "histogram" represents the total number of occurrences, m, of a class of pixels in a window of size L. From this, a normalized histogram is formed, which yields an estimate of the probability density function, P(m,L), for each window size L.

All selected moments (q) of the P(m,L) distributions for each of the desired window sizes are determined, and a linear regression or "best fit line" is found for the moment generating function, log(M(L))^1/q, versus the log of the window size, log(L). The slope of the "best fit line" provides an estimate for the fractal dimension, D.

The center pixel of the largest window, Lmax, is replaced with the computed fractal dimension based on the probability density function, P(m,L). The largest window size, Lmax, determines the resulting size of the fractal dimension image. This results in a border of size (Lmax / 2) around the image, which is set to zero.

The input arguments are described as follows:

-i specifies the input image, which must be of data type BYTE or INTEGER. The input image must be a single band image.

-o1 specifies the output image, which will be a multiband image representing the fractal dimension of the input image for each of the specified moments. The resulting output image will be of data storage type FLOAT, and will have a border of pixels of value zero. The size of the border will be determined by the size of the largest window specified. This can be determined from the following formulation, (Lmax -1) / 2 = border size. Where Lmax is the size of the largest window used. For example, if the largest window size is 9, then the border size will be (9 - 1) / 2 = 4 pixels.

-o2 specifies fractal dimension image, representing the fractal dimension vectors for each class and moment. This image will always have a row size of 1 and a column size determined by the number of classes in the input image. The number of data bands will be equal to the number of moments used in the estimation of the fractal dimension.

-f1 uses an ASCII file as output for the image information specifying the size of the input image, number of classes in the input image, and the range of moments used in the estimation of the fractal dimension. Also included is a listing of the fractal dimensions for each class and moment.

-l specifies the initial or lower size of the sliding window. This MUST be an odd number resulting in a window with a center pixel. This means that the minimum size of the sliding window is 3 x 3. The default value is 3, resulting in an initial window size of 3 x 3.

-u specifies the final or upper size of the sliding window. This MUST also be an odd number resulting in a window with a center pixel. The size of the upper window MUST be at least one step greater than the lower window size. That is, if the lower window is 3 x 3, then the minimum size of the upper window must be 5 x 5, or one step greater than the lower window. This is necessary, since there must be at least two points for a best fit line to be formed determining the slope and ultimately the fractal dimension.

-s specifies the step size or interval used when specifying a range of window sizes. The default value is two, corresponding to the next odd window size with a center pixel. This may be helpful when a wide range of window sizes is required, as it will reduce the number of points generated for the best fit line and hence the number of calculations.

-q specifies the range of moments to base the P(m,L) fractal dimension calculations on. This must be an odd number, corresponding to the number of moments centered on zero. For example, a value of 5 would result in the computation of the fractal dimension for the range of moments, -2, -1, 0, 1, 2. The default value is 3, resulting in the range of -1, 0, 1.

The input image MUST be of data storage type BYTE or INTEGER.

Examples

vpml -i in_img -o1 out_img1 -o2 out_img2 -f1 file1 -u 11 -s 4 -q 5

this will estimate the fractal dimension of the in_img using a range of sliding window sizes from 3 x 3 (default lower value of 3) to an upper size of 11 x 11, with a step size of 4. This means that windows of size 3 x 3, 7 x 7, and 11 x 11 will be used in the fractal dimension calculation. A value of 5 was selected for the number of moments to be calculated, meaning that the fractal dimension will be computed for the range of moments, -2, -1, 0, 1, 2. The resulting output image, out_img1 will consist of a five-band FLOAT image with each band representing the fractal dimension for a particular moment, q. The out_img2 will consist of a single row, five-band image specifying the fractal dimension for each class of the input image and moment, q. The ASCII file1 will provide the user with information on the size of the input image, number of classes, and the resulting fractal dimensions for each class and moment.

See Also
vfractal(1)

Restrictions

The input image MUST be of data storage type BYTE or INTEGER. The output images are of data storage type FLOAT.

References
A reference for the pml algorithm is p. 67 of The Science of Fract. Images edited by Hienz-Otto Peitgen and Dietmar Saupe.

Command Line Arguments

```
Usage for vpml: Estimate Fractal Dimension of Image Based on P(m,L)  (K1)
% vpml
  -i          (infile)  input image
  -o1         (outfile) resulting multiband fractal dimension output image
  -o2         (outfile) output image specifying fractal dimension of each class

  [-l]        (integer) select the initial size of the sliding window
              (3 <= l <= 10000, default = 3)
  [-u]        (integer) select the final size of the sliding window
              (5 <= u <= 10000, default = 5)
  [-s]        (integer) select the window step size or interval
              (2 <= s <= 10000, default = 2)
  [-q]        (integer) select the range of moments (-r/2 <= 0 >= r/2)
              (3 <= q <= 1000, default = 3)
  [-f1]       (outfile) output file for P(m,L) statistics
              (default = {none})
```

B.39. vqerr — *Compute Error between Quantized Image and Original (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Classification*
Operator:	*Quantization Error*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description
vqerr computes the RMS quantization error between a quantized image and the original image.

The first input image is the quantized image and should have maps attached. The data type of the first image should be BYTE or INT. The map should be of the same data type as the data in the second image (the original). The second (original) image should have no maps attached.

The number of columns in the map of the first input image should match the number of bands in the second (original) input image.

Examples

% igamut -i mandril.rgb.xv -o mandril.quant & kformats -i mandril.quant -o mandril.quant -xvimage % vqerr -i1 mandril.quant -i2 mandril.rgb.xv -o mandril.new

will quantize the color mandril image and compute the quantization error.

See Also

vipl(3),vquant(1),igamut

Restrictions

vqerr will not operate on BIT, COMPLEX, or DOUBLE data storage types.

Command Line Arguments

```
Usage for vqerr: Compute Error between Quantized Image and Original  (K1)
% vqerr
  -i1        (infile)  Quantized image
  -i2        (infile)  Original image
  -o         (outfile) Output ASCII File

  [-m]       (infile)  operation masking image
             (default = {none})
```

B.40. vqmed — *Median Filter the Image Using Quicksort to Find Median (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Spatial Filters*
Operator:	*Median Quicksort*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

NOTE: Program will NOT execute! vqmed functionality is replaced by IMAGE::imedian. Command line arguments have changed slightly. The -w option is now -wsize, and the -h option is now -hsize.

vqmed computes a two dimensional median filter of width w and height h over the given image, img. The median is computed by moving a w by h pixel mask over the image and sorting the masked values. The value of the median filter is the (mask size+1)/2 th value in the sorted array.

The resulting image is of the same type as the input image.

Examples

vqmed -i input.image -o output.image -w 3 -h 3

This command performs a median filter of width 3 and height 3 on input.image and stores the result in output.image.

See Also

IMAGE::imedian RETRO::vhmed

Restrictions

vqmed will not operate on data storage type complex.

Command Line Arguments

```
Usage for vqmed: Median Filter the Image Using Quicksort to Find Median  (K1)
% vqmed
  -i         (infile)  input image
  -o         (outfile) resulting output image

  [-w]       (integer) filter width
             (1 <= w <= 32, default = 3)
  [-h]       (integer) filter height
             (1 <= h <= 32, default = 3)
```

B.41. vquant — *Perform N-Dimensional Vector Quantization or Classification (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Classification*
Operator:	*Non-Parametric*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vquant performs N-dimensional vector quantization, also known as non-parametric vector classification. The technique is based on Paul Heckbert's median cut, with the modification that the histograms may have many independent axes, and are stored in a sparse data structure. Additionally, the splitting of a cluster is governed by the 2-norm of the subspace spanned by that cluster. That is, the decision regarding which "box" to split is made by finding the box representing the largest subspace in the 2-norm sense.

When splitting a cluster, the split axis is normally chosen to be the axis with the largest span across the enclosed data points, and the split point is the middle of the span. By setting the variance split flag, the split axis is chosen to be the axis with the largest variance of data across that axis, which should give better results at the cost of more computation. When the variance split flag is selected, the split point is at the mean of the distribution of data long the split axis.

References:

P. Heckbert, "Color Image Quantization for Frame Buffer Display", Computer Graphics, Vol. 16, No. 3, July 1982, p297.

The output from vquant is compatible with the *spectrum* classification application when the -m (map enable) option is set to true. The resulting cluster number image will have the ispare1 and ispare2 fields set. Ispare1 will be set to 0 and ispare2 will be set to the dimension of the vector.

Examples

vquant -i satellite_multiband -o classified -n 128

will generate a single-band output image, with pixel values between 0 and 127, with a mapping scheme of VFF_MS_ONEPERBAND, where the number of rows in the map is 128 and the number of columns is the number of planes in the input image. The actual coordinates of the vector representatives are stored as rows in the map.

Restrictions

vquant will currently only work on images of type VFF_TYP_FLOAT. Additionally, the output is currently restricted to type VFF_TYP_1_BYTE, and this in turn restricts the maximum number of representative vectors to 256.

Additionally, vquant suffers from the same statistical influences that igamut does; see igamut for more details.

Command Line Arguments

```
Usage for vquant: Perform N-Dimensional Vector Quantization or Classification (K1)
% vquant
  -i         (infile)  Input image
  -o         (outfile) Output cluster number image

  [-n]       (integer) Number of vectors
             (n > 0, default = 128)
  [-m]       (boolean) Output Image Map Enable
             (default = true)
  [-s]       (boolean) Select split point:  false = mid-span, true = mean
             (default = false)
  [-axis]    (integer) Select splitting axis
          1  (Maximum span axis),
          2  (Maximum variance axis),
          or 3  (Principal eigenvector)
[default: 1]
  [-o1]      (outfile) Output cluster center image
             (default = {none})
  [-o2]      (outfile) Output cluster variance image
             (default = {none})
```

B.42. vsamp — *Sample an Image Along an Arbitrary Line (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Size Operators*
Operator:	*Sample on Line*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

NOTE: Program will NOT execute! vsamp functionality is replaced by the DATAMANIP::ksampline

operator. The following command line arguments have changed.
-x1 (starting x coordinate for line), is now -wc1
-x2 (ending x coordinate for line), is now -wc2
-y1 (starting y coordinate for line), is now -hc1
-y2 (ending y coordinate for line), is now -hc1
the values may be be slightly different because vsamp used bilinear interpolation whereas ksampline uses inverse-distance weighting interpolation (which is mush easier to extend to 5D and mask'd data).

Given an image of arbitrary size and a line through that image, sample the image along the line. The sampling interval is independent of image size and line orientation. The value at each sample point is obtained by bilinear interpolation.

The first sample point is precisely at the starting point and the final sample is precisely at the ending point.

Examples

vsamp -i diffract.xv -o line_data.xv -x1 0 -y1 0 -x2 0 -y2 0 -n 400

Will sample the image diffract.xv to obtain a one dimensional data array in line_data.xv. The line goes from the top left corner to the bottom right. 400 evenly spaced samples will be obtained via bilinear interpolation.

See Also

DATAMANIP::ksampline

Restrictions

vsamp will only work on images of type FLOAT.

Command Line Arguments

```
Usage for vsamp: Sample an Image Along an Arbitrary Line  (K1)
% vsamp
  -i         (infile)  input image
  -o         (outfile) output image

  [-x1]      (integer) starting x coordinate for line
             (x1 >= 0, default = 0)
  [-x2]      (integer) ending x coordinate for line
             (x2 >= 0, default = 511)
  [-y1]      (integer) starting y coordinate for line
             (y1 >= 0, default = 255)
  [-y2]      (integer) ending y coordinate for line
             (y2 >= 0, default = 255)
  [-n]       (integer) number of samples
             (n > 0, default = 1)
```

B.43. vsdef — *Second Derivative Operator for Symmetric Exponential Filter (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Spatial Filters*
Operator:	*SDEF Edge Extract*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

This Algorithm has been written by the team of:

Professor SERGE CASTAN

IRIT, URA 1399

118, route de Narbonne 31062 Toulouse FRANCE

OPTIMAL FILTER(ISEF) FOR EDGE DETECTION

PRINCIPLE :

I. Introduction

Edge detection is one of the most important subjects in image processing, which finds wide applications in the pattern recognition, the scene analysis and the 3-D vision, because the edges correspond in general to the important changes of physical or geometrical properties of objects in the scene and they are widely used as primitives in the pattern recognition, the image matching etc.

The edges coincide, generally speaking, grey level transition, they can be detected by maxima of gradient or the zero-crossing of the second derivatives calculated by some differential operators. Because the differential operators are sensitive to noise, a preprocessing such as smoothing is often necessary to eliminate the noise. A well-known smoothing filter is Gaussian filter and the edges can therefore be detected by a Laplacian-Gaussian filter. But there is an essential difficulty of the Laplacian-Gaussian filter which is the contradiction between the smoothing effect and the precision of edge localization. To overcome this difficulty, We proposed the optimal linear filter based on one step model (a step edge and the white noise) and the multi-edge model [9][10][11]. This optimal smoothing filter is a symmetric exponential filter of an infinitely large window size and can be realized by very simple recursive algorithm. It is proved that the band limited Laplacian of an input image filtered by this filter can be calculated from the Difference between the input and the output of this Recursive Filter (DRF). The edges detected by DRF method are less noisy and with a much better precision of localization.

The maxima of gradient or zeros of the second directional derivative along the gradient are a natural definition of intensity edges. Zeros of the Laplacian are only extensively used for their computational convenience. However, we must stress here that the zeros crossing of the Laplacian are not always coincided with the maxima of gradient, for example, the zeros of the Laplacian are farther apart than the maxima of gradient for circularly symmetric patterns, this lack of localization by the Laplacian can also be seen in the fact that zeros of Laplacian "swing wide" of corners. Therefore, it had better to detect the edges by maxima of gradient or zeros of the second directional derivative along the gradient .

So we propose two methods for edges detection, one uses maxima of Gradient (GEF), another uses the zeros crossing of Second directional Derivative along the gradient (SDEF).

II. The First and the Second Directional Derivative Operators for Symmetric Exponential Filter

A normalized symmetric exponential filter on 1-D can be written :

fL(x)=C*a0*(1-a0)**|x|=f1(x)#f2(x)=C*(f1(x)+f2(x)-a0*d(x)) (1)

where:

C=1/(2-a0), # means the convolution, d(x) is dirac function.

f1(x)={0 if x<0 and a0*(1-a0)**x if x>= 0 (2)

f2(x)={0 if x>0 and a0*(1-a0)**-x if x<= 0 (2)

we can write the first derivative operator of exponential filter:

fL'(x)=f2(x)-f1(x) (3)

And we can obtain the normalized second derivative operator of exponential filter :

$$fL''(x)=f1(x)+f2(x)-2*d(x) \quad (4)$$

Because the exponential function is separable, we can write out 2-D exponential filter:

$$f(x,y) = fL(x)*fL(y) \quad (5)$$

From the equations (18),(19) and (20), the first and the second directional derivative operators for symmetric exponential filter can be written like this:

$$fx(x,y) = fL(y)*(f2(x)-f1(x)) \quad (6)$$

$$fy(x,y) = fL(x)*(f2(y)-f1(y)) \quad (7)$$

$$fxx(x,y) = fL(y)*(f1(x)+f2(x)-2*d(x)) \quad (8)$$

$$fyy(x,y) = fL(x)*(f1(y)+f2(y)-2*d(y)) \quad (9)$$

III. The Recursive Algorithm for realizing the These Directional Derivative Operators of Symmetric Exponential Filter

The exponential filter is an IIR filter corresponding to an infinite window size, so we should realize the functions f1(x) and f2(x) (see formula (2)) by a recursive algorithm.

Supposing I(x,y) is the input image,
I1(x,y)=I(x,y)#f1(x) and I2(x,y)=I(x,y)#f2(x),
we have the recursive algorithm :

I1(x,y)=I1(x-1,y)+a0*(I(x,y)-I1(x-1,y)) (10)

I2(x,y)=I2(x+1,y)+a0*(I(x,y)-I2(x+1,y))

From the equations (1),(6),(7),(8) and (9), the band-limited first and second directional derivative of input image can be calculated by the recursive algorithm f1 and f2 as follows

Ix(x,y)=I(x,y)#f1(y)#f2(y)#(f2(x)-f1(x)) (11)

Ixx(x,y)=I(x,y)#f1(y)#f2(y)#(f2(x)+f1(x))-2*I(x,y)#f1(y)#f2(y) (12)

Iy(x,y) = I(x,y)#f1(x)#f2(x)#(f2(y)-f1(y)) (13)

Iyy(x,y) = I(x,y)#f1(x)#f2(x)#(f2(y)+f1(y))-2*I(x,y)#f1(x)#f2(x) (14)

With this algorithm, we can calculate at the same time the band-limited first and second directional derivative Ix and Ixx (or Iy and Iyy) of input image.

IV. Edges Detection

The band-limited first and second directional derivative of input image can be obtained by the algorithms as stated above. Using them, we can then realize the edge detection for an image.

The maxima of gradient or zeros of the second directional derivative along the gradient are a natural way of characterizing and localizing intensity edges, so we present here to detect the edges from the maxima of gradient or zeros of the second directional derivative along the gradient by using the differential operators of exponential filter.

1. Edges from the maxima of gradient

Using the first directional derivative operator of the exponential filter, the two band-limited first directional derivatives Ix and Iy can be calculated, and the gradient vector can be therefore determined approximatively for every point in image. The gradient magnitude image is then non maxima suppressed in the gradient direction and thresholded with hysteresis, i.e. if the entire segment of the contour lies above a low threshold T1, and at least one of part of which is above a high threshold T2, that contour is output. The non maxima suppression scheme requires three points, one of which will be the current point, and the other two should be estimated of the gradient magnitude at points displaced from the current point by vector normal to the edge direction.

2. Edges from the zero-crossings of the second directional derivative along the gradient direction

Because edges detected from local gradient maxima can not be a pixel width (less good precision of localization), we propose an other method which detect the edges from the zeros crossing of the second directional derivative along the gradient direction.

We can calculate Ix,Iy,Ixx and Iyy by using the method shown in paragraph III, and therefore obtain approximatively the gradient vector and the second derivative in the gradient direction for every point in image. We extract at first the zero crossing of second derivative along the gradient direction on which the gradient magnitude must be above a low threshold, so an edges image is obtained. To this image, the entire segment of the contour will be kept, if the gradient magnitude on at least one part of this contour is above a high threshold.

V. Comparison of Performance of the Filters

Filtering is a problem of estimation from noisy signal, and edge detection is a problem of estimating the position of maximal local signal change. Up to now, many works are done for edges detection in image, and different filters are proposed, for example, Gaussian filter, Canny filter, exponential filter, Deriche filter etc...

We appreciate the performance of the filters as follows :

(1) Precision of edge localization

According to our analysis [10], we can calculate the average localization error xe for Gaussian filter, Canny filter [15], Deriche filter [16] and the exponential filter :

xeG = (4*(2*e*3.14)**0.5)/a

xec = 0.81/a

xeD = 4*exp(-1)/a = 1.47/ a

xeE = 0

i.e. xeG > xeD > xec > xeE = 0.

So, we can see that the exponential filter localizes edge points with the best precision.

(2) Signal/Noise ratio on the edge point detected

Because xe is the average estimation for the position of the edge point detected, we propose to calculate Signal/Noise ratio (Eq.(7)) at the point xe.

And the signal/noise ratio for the Gaussian filter, Canny filter, Deriche filter and the exponential filter is :

SNRG = 2*s*exp(-32*s)/ ((3.14)**0.5)

SNRc = 0.39/ a

SNRD = 0.64/ a

SNRE =1/ a

i.e. SNRE > SNRD > SNRc > SNRG.

Then, we see that the exponential filter has the best noise eliminating effect among the above four.

(3) Complexity of calculation

For the complexity of calculation, we only tell the difference from exponential filter and Deriche filter [16], because they are implemented by recursive algorithms which have a simpler calculation.

Because ISEF can be realized by first order recursive filter, the ISEF algorithms are much simpler than that of Deriche filter. Besides, the ISEF algorithms can be implemented independently to every line and every column, it can be easily realized by a parallel system.

According to the analysis results above, the ISEF filter is superior to the others at the 3 principal aspects of the performance of the filter.

VI. Conclusion

The symmetric exponential filter of an infinite large window size is an optimal linear filter deduced from one step edge model and the multi-edge model, now we further prove that the symmetric exponential filter is the optimal edge detection filter in the criteria of the signal to noise ratio, the localization precision and unique maximum. Obviously, the real images will be still more complicated than these models, however DRF method has already provided good results for different type of images. The results obtained through the two new methods further show the superior performance of this filter. The theoretical analysis for the performance of the filters shows also that the exponential filter is superior to the other current filters.

The first and second directional derivative operators can be realized by recursive algorithm and calculated at the same time. The new algorithms are therefore very simple as well as DRF algorithm, and they are also easy to implemented in a parallel way.

Examples

vsdef -i cross.xv -o output.xv

will compute a edge extraction by using the first directional derivative operator for symmetric exponential filter on image cross.xv and write the result in output.xv

For the SDEF edge extraction the filter parameter value is generally a0 between 0.40 and 0.9 (0.45 [default] is a good value)

The hysteresis Thresholds T1 and T2 can be set to 12 and 15.

The minimum Length of segment is up to the user. It is set by default to 10.

-man1_long_description_end

-man1_examples vsdef -i cross.xv -o output.xv

will compute an edge extraction by using the second directional derivative operator for symmetric exponential filter on image cross.xv and write the result in output.xv

For the SDEF edge extraction the filter parameter value is generally a0 between 0.40 and 0.9 (0.45 [default] is a good value)

The hysteresis Thresholds T1 and T2 can be set to 3 and 4. (12 and 15 seem to be a set of good values as well, it depend on the level of noise in the image.)

The minimum Length of segment is up to the user. It is set by default to 10.

See Also

vdrf(1), lvdrf(3), vgef(1), lvgef(3).

Restrictions

works only with single band byte images.

Note that this routine was converted directly from Khoros 1.0. Due to the mixing of unsigned versus signed char, and the difference in the rules that resolve the mixing of these data types between ANSI C and the original K&R C that the code was written in, the results of running this program on an image will not be the same as with vdrf in Khoros 1.0. There is really no way to resolve which version is more correct, but the ANSI C version should at least be more consistent across machine architectures.

References

[1] W.K. PRATT, Digital Image Processing, New York, 1978.

[2] J. PREWITT, Object Enhancement and Extraction,Picture Processing and Psychopictories, Etd. by B. Lipkin and A. Rosenfeld, New York, pp.75-149, 1970.

[3] M. HUECKEL, An Operator Which Locates Edges in Digitized Pictures. J.A.C.M., Vol. 18, pp 113-125, 1971.

[4] R.O. DUDA and P.E. HART, Pattern Classification and Scene Analysis. Wiley, New York, 1973.

[5] R. HARALICK,Edge and Region Analysis for Digital Image Data. C.G.I.P., Vol. 12, pp 60-73, 1980.

[6] R.HARALICK and L.WATSON, A Facet Model for Image Data. C.G.I.P., Vol. 15, pp 113-129, 1981.

[8] D. MARR and E.C. HILDRETH, Theory of Edge Detection. Proc. R. Soc. Lond. B, Vol. 207, pp 187-217, 1980.

[9] J. SHEN and S. CASTAN, Un nouvel algorithme de detection de contours, proceedings of 5th Conf. on P.R.&.A.I. (in French), Grenoble, 1985.

[10] J. SHEN and S. CASTAN, An Optimal Linear Operator for Edge Detection. Proc. CVPR'86, Miami,1986.

[11] J. SHEN and S. CASTAN, Edge Detection Based on Multi-Edge Models.Proc. SPIE'87, Cannes, 1987.

[12] J. SHEN and S. CASTAN, Further Results on DRF Method for Edge Detection. 9th I.C.P.R., ROME, 1988.

[13] V.TORRE and T.A.POGGIO, On Edge Detection IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. Pami-8, N 2, March 1986.

[14] J.S.CHEN and G.MEDIONI, Detection, Localization, and Estimation of Edges. IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. 11, N 2, February 1989.

[15] J.F.CANNY, Finding Edges And Lines in Images. MIT Technical Report N 720, 1983.

[16] R. DERICHE, Optimal Edge Detection Using Recursive Filtering. In proc. First International Conference on Computer Vision, London, June 8-12 1987.

Command Line Arguments

```
Usage for vsdef: Second Derivative Operator for Symmetric Exponential Filter  (K1)
% vsdef
  -i         (infile)  Input Image
  -o         (outfile) Output Image

  [-a0]      (float)   Exponential Filter Parameter for GEF
             (0 <= a0 <= 1, default = 0.45)
  [-t1]      (integer) 1st Hysteresis Threshold
             (0 <= t1 <= 255, default = 3)
  [-t2]      (integer) 2nd Hysteresis Threshold
             (0 <= t2 <= 255, default = 4)
  [-l]       (integer) Minimum Number of Pixels in a Segment
             (0 <= l <= 100000, default = 10)
```

B.44. vshape — *Perform Shape Analysis on Image (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Feature Extraction*
Operator:	*Shape Analysis*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vshape performs a shape analysis on two different types of images, by extracting features like moments, area, centroids, etc.

The input images can be:

1. A labeled or clustered image. The grey level of a pixel in a labeled or clustered image is specifically used to assign a pixel to a particular labeled region. For a labeled image (output of vlabel routine), the classes or regions are connected, however this is not true for a cluster number image (output of vquant, or vkmeans). This difference can confuse a novice user, because vshape will not differentiate between connected or non-connected regions.

2. A grey level image. In this particular case, the features will be computed for the entire image. Some of these features may not appear to make sense, nevertheless they will still contribute significant information to the regions.

Before providing more information on vshape, the requirements for the input image are:

- The data storage type can range from VFF_TYP_1_BYTE to VFF_TYP_FLOAT.
- The number of bands MUST be 1.
- The map enable MUST be VFF_MAP_OPTIONAL.

Description of vshape outputs.

The features listed below are computed for each region or for the entire image, depending on the input image.

- **Centroid** on x and y axis.
- **Variances** on x and y.
- **Area** or **weighted area**.
- **Orientation** of the principal axis.
- **Eccentricity** of the shape.
- **Bounded box** coordinates.

The following features are optional.

- **Standard Moments**.
- **Central Moments** and **Normalized Central Moments**.
- **Invariant Moments**.

These features are provided in an optional ASCII file.

As mentioned above, the centroid coordinates of the grey level input image provides more information than the general sense of the centroid of a region.

In addition to the ASCII file, three other output files can be generated.

1. The Axis Image. This image contains a representation of the moments of the regions contained in the input image. It can only be generated when the input image is a not a grey level image. There are two possibilities for representing the moment.

   The first one is an image containing a cross for each region. The cross center is located at the centroid of the region and the horizontal and vertical axes stop outside of the bounded box of the region.

   The second one is an image still containing a cross for each region but this time the cross is representative of the moment value in the principal axis system. The principal axis is the coordinate system in which the cross moment, m11 (see reference for theory on moment below), is equal to zero. Therefore, this cross forms an angle (theta) with the x axis, and its length in both directions are proportional to the "length" of the shape is that direction.

2. The Outline Image. This image contains the outlines of all the regions contained in this image. Again, this can only be generated if this input image is not a grey level image.

3. The VIFF Image. The image contains all of the statistics. The VIFF output file has 61 (number of statistics) bands and the number of vectors in this file depends on the number of regions specified in the input image. If the statistics flags are not set, the corresponding statistics will default to zero.

For these three output images there is a final possibility, which is to assign all outlines or crosses a grey level of 255, or their previous region number. This enables the user to visualize the input image, the outline image, and the axis image using pseudo color to assign a different color to each region. This aids in interpreting the regions, and yields an artistic view of the image.

The options to vshape are listed below:

-i1 cluster number or labeled image. The region number zero (the label or cluster number that equals zero) is ignored because it is non-representative or is due to a border requirement from vlabel, vquant, vkmeans, etc.

-i2 grey_level image file. This option is mutually exclusive with -i1. Using -i1 will cancel the generation of outline and axis images even if they were specified.

-o2 outline image. This image contains the outline of every region in the image except region 0. If the color of the outline was specified as white (255), the outline thickness will be two pixels wide. This is due to the fact that if region n and region m are next to each other, the outline of n will belong to the pixels of n that have at least one neighbor not belonging to n. The same principle applies to region m, and results in an edge between regions n and m comprised of a pixel belonging to n and one belonging to m. When using a different color for each outline, this phenomenon doesn't occur.

-o3 viff format output file. If the -o3, VIFF output file, option is selected, all of the statistics will be stored in the VIFF output file. If the statistics flags are not set, the corresponding statistics will default to zero.

Currently there are 61 statistics bands:

0. surf_image: Number of pixels in the entire image.
1. area: Number of pixels in the object = m00.
2. percent_area: Ratio area / surf_image.
3. real_area: Area in square meters.
4. xcent: x_cent of object.
5. ycent: y_cent of object.
6. sigma_x: variance = sqrt [mu20 / m00].
7. sigma_y: variance = sqrt [mu02 / m00].
8. sigma_MX: sqrt [MX / m00].
9. sigma_MY: sqrt [MY / m00].
10. theta: angle between the major axis of the shape.
11. eccentricity: Measures the eccentricity of the figure.

12. m00: STANDARD MOMENTS
13. m01:
14. m10:
15. m11:
16. m20:
17. m02:
18. m21:
19. m12:
20. m03:
21. m30:

22. mu00: CENTRAL MOMENTS
23. mu10:
24. mu01:
25. mu20:
26. mu02:
27. mu11:
28. mu30:
29. mu03:
30. mu21:
31. mu12:

32. nu00: NORMALIZED CENTRAL MOMENTS
33. nu10:
34. nu01:
35. nu20:
36. nu02:
37. nu11:
38. nu30:
39. nu03:
40. nu21:
41. nu12:

42. M1: INVARIANT MOMENTS
43. M2:
44. M3:
45. M4:
46. M5:
47. M6:
48. M7:

49. right: gives the rightmost point of the object
50. left: gives the leftmost point of the object
51. top: gives the uppermost point of the object
52. bottom: gives the lowest point of the object
53. axis_x1deb: moment on principal axis 1.
54. axis_x1end: moment on principal axis 1.
55. axis_y1deb: moment on principal axis 1.
56. axis_y1end: moment on principal axis 1.
57. axis_x2deb: moment on principal axis 2.
58. axis_x2end: moment on principal axis 2.
59. axis_y2deb: moment on principal axis 2.
60. axis_y2end: moment on principal axis 2.

-f name of ASCII file (record of feature values for every region, only the specified features will be written to this file).

-ms when set to one, this computes the set of standard moments and writes them to the ASCII output file, providing the ASCII output file was specified.

-mc when set to one, this computes the set of central moments and normalized central moments. The central moments are invariant to translation, and the normalized central moments are invariant to both size and translation.

-mi when set to one, this computes the set of invariant moments. This set is invariant to size, position (translation), rotation, and reflection. (Only the last moment, M7, isn't invariant to reflection, which can be used to discriminate an object from its reflection in a mirror). This set can be used on a grey level image of a particular object and can be sent as input to a pattern recognition algorithm.

-mr when set to 0, the axis image will contain crosses with horizontal and vertical axes inside of the bounded box of the region. When set to 1, the axes are along the principal axis (giving the general orientation and length of the region).

-col when set to 0, the outline and axis images, if specified, will be black and white images. When set to 1, the background will be black 0, and each cross or outline will take on the "color" of that region.

The output images are of data storage type VFF_4_BYTE, to allow a sufficient number of distinct regions in the image.

References:

1. Vision In Man And Machine by Martin D. Levine. Chapter 10. "Shape". Page 480.

2. Digital Image Processing by R.C. Gonzales & P. Wintz. Chapter "Representation and description". Page 418.

3. IRE Transactions on information theory,vol IT-8,1962 "Visual Pattern Recognition by Moment Invariants" by Ming_Kuei Hu. Page 179.

Examples

vshape -i1 image 1 -o1 image2 -o2 image3 -o3 image4 -f file -ms 1 -mc 0 -mi 1 -mr 1 -col 1

This example will produce an ASCII file containing the information for each region, and an outline and principal axis representation image, with a color equal to the region number. The information will contain general features and the set of standard and invariant moments but not the central set, in ASCII file format and in viff file format.

Restrictions

vshape will not work with an image which type is:

VFF_TYP_COMPLEX

VFF_TYP_DOUBLE

Command Line Arguments

```
Usage for vshape: Perform Shape Analysis on Image  (K1)
% vshape

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i1] (infile) Input Cluster/Label Image
          (default = {none})
    [-i2] (infile) Input Grey Level Image
          (default = {none})

  [-ms]       (boolean) if true, compute standard moments
              (default = false)
  [-mc]       (boolean) if true, compute central moments
              (default = false)
  [-mi]       (boolean) if true, compute invariant moments
              (default = false)
  [-mr]       (boolean) if false, computes axis of bounded box
              (default = false)
  [-col]      (boolean) if false, color = white, with black background
              (default = false)
  [-o1]       (outfile) output Axis image
              (default = {none})
  [-o2]       (outfile) output outline image
              (default = {none})
  [-o3]       (outfile) output VIFF file containing statistics
              (default = {none})
  [-f]        (outfile) output ASCII file
              (default = {none})
```

B.45. vslope — *Compute Slope and Aspect Images from Elevation Data (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Surface*
Operator:	*Slopes and Aspects*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vslope computes the slope and aspect images from an input elevation data.

The slope is always calculated as a positive number and represents the slope in the direction of the gradient at that point. The direction in which the slope is calculated depends only on the direction of the steepest gradient at the point. The slope may be computed in three forms: degrees, radians, and percent rise. Degrees and radians are used to measure the angle between a tangent to the surface at the point the slope is being calculated and a horizontal reference plane. Percent rise is a measure of how much the tangent rises with respect to the horizontal distance. For instance, if the tangent at a point rises 50 meters for every 100 meters horizontally, the slope is 50 percent. The form used for the output slopes is selected using the command line argument '-so'.

The aspect of the elevation image tells which direction the slope is facing. The aspect is reported in the direction of the increasing slope. The aspect may also be calculated in three forms: degrees, radians, and quadrants. Degrees and radians give the clockwise angle between the positive x-axis (right to left on the image and east in geographical terms) and the direction the slope is facing. For instance, if the slope is facing the top of the image (north) then the aspect will be 90 degrees or pi/2 radians. When reported in quadrants, the aspect is given a value that depends on the range the direction it faces. The circle is divided into 24 regions of 15 degrees each, and the region that the aspect falls in determines the value that will be assigned to it. North is always assumed to be at the top of the image, and East is always to the right. A table of the quadrant values is given below:

Aspect Value	Range in Degrees	Description
1	353 - 7	east facing
2	8 - 22	15 degrees north of east
3	23 - 37	degrees north of east
4	38 - 52	northeast facing
5	53 - 67	30 degrees east of north
6	68 - 82	15 degrees east of north
7	83 - 97	north facing
8	98 - 112	15 degrees west of north
9	113 - 127	30 degrees west of north
10	128 - 142	northwest facing
11	143 - 157	30 degrees north of west
12	158 - 172	15 degrees north of west
13	173 - 187	west facing
14	188 - 202	15 degrees south of west
15	203 - 217	30 degrees south of west
16	218 - 232	southwest facing
17	233 - 247	30 degrees west of south
18	248 - 262	15 degrees west of south
19	263 - 277	south facing
20	278 - 292	15 degrees east of south
21	293 - 307	30 degrees east of south
22	308 - 322	southeast facing
23	323 - 337	30 degrees south of east
24	338 - 352	15 degrees south of east
25	no aspect (flat)	

The form of the output aspect is selected using the '-to' command line argument. When the surface has no slope, the aspect is undefined. The value that will be assigned for an undefined aspect is set using the '-z' option on the command line.

The input elevation file must be in viff format, and may be of type byte (VFF_TYP_1_BYTE), short (VFF_TYP_2_BYTE), int (VFF_TYP_4_BYTE), or float (VFF_TYP_FLOAT). The input data will automatically be converted to float, and both the slope and aspect output files will always be type float regardless of the input data type. Any type of map that is not forced (VFF_MAP_FORCE) is allowed on the input file, and the map will be transferred as is to the output. Explicit location data is not allowed in the input file, and will result in an error. The pixels in the input file are assumed to be evenly sampled where the sampling interval is given in the 'pixsizx' and 'pixsizy' fields in the viff header. The units used for the sampling interval are arbitrary but must be the same units that the elevations in the file use. The sampling interval is not allowed to be zero. Finally, the input file is restricted to a single image.

Examples

vslope -i elevation.image -s slope.image -so 2 -t aspect.image -to 2

This command computes both the slope and the aspect images from the input elevation image. "Elevation.image" is a float type image that is evenly sampled. "Slope.image" is the output slope image. It is type float, and the -so 2 option specifies that the slopes will be given in radians. "Aspect.image" is the output aspect image. It is also type float, and the -to 2 option specifies that the aspects will be given in radians.

See Also

vsurf(1)

Restrictions

vslope works on input images with data types byte (VFF_TYP_1_BYTE), short (VFF_TYP_2_BYTE), int (VFF_TYP_4_BYTE), and float (VFF_TYP_FLOAT), but the type of the output image will always be float regardless of the input type. Maps on the input image will be transferred directly to the output image, but forced maps (VFF_MAP_FORCE) are not accepted. Explicit location data is not accepted. The input image is restricted to one image per file.

Command Line Arguments

```
Usage for vslope: Compute Slope and Aspect Images from Elevation Data  (K1)
% vslope
  -i         (infile)  input image

  [-so]      (integer) slope options
          0  (Output Slope in Degrees),
          1  (Output Slope in Radians),
          or 2  (Output Slope in Percent Rise)
[default: 0]
  [-to]      (integer) aspect options
          0  (Output Aspect in Degrees),
          1  (Output Aspect in Radians),
          or 2  (Output Aspect in Quadrants)
[default: 0]
  [-z]       (float)   Value for Regions With no Aspect (flat)
             (unbounded, default = 0)
  [-s]       (outfile) output slope image
             (default = {none})
  [-t]       (outfile) output aspect image
             (default = {none})
```

B.46. vspatial — *Compute Spatial Features Using NxM Window (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Feature Extraction*
Operator:	*Spatial Analysis*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vspatial performs a spatial feature extraction on an image using a selected statistical operation. The statistical operation is computed, using overlapping windows, over the entire image. The center pixel is updated in each window based on the calculated statistics of that window.

The input image, which must be of data storage type BYTE, corresponds to the -i argument. The output image, which corresponds to the -o argument, is stored as data type FLOAT. The output image may be a multiband image depending on how many spatial operators were selected. The resulting image will have a border of pixels of value 0, since the window operation does not calculate values for border pixels.

The width and height of the window is specified by the -w and -h window width and height arguments. The width and height arguments cannot be an even number. The window dimensions must be odd (ie. 3x3, 5x5, 7x7, etc.), since the center pixel is replaced with the computed value of the window data. The default window size is 3x3.

Each pixel in the image is updated with a computed value except the border pixels. A 3x3 window will result in a border of one pixel, a 5x5 window will result in a border of two pixels, etc.

There are six statistical operations that may be selected. Any combination of the statistical operations may be selected by toggling the desired argument on (ie. argument followed by a 1). The default for each argument is unselected (ie. argument followed by a 0), except -m (mean), which is selected. The six possible statistical operations and their arguments are:

-m -- calculates the mean of the image.
-v -- calculates the variance of the image.
-c -- calculates the contrast of the image.
-s -- calculates the angular second moment of the image.
-e -- calculates the entropy of the image.
-d -- calculates the dispersion of the image.

The resulting feature vector may contain from one to six bands of data, depending upon the number of arguments selected.

All input images must be of data storage type BYTE.

All output images are of data storage type FLOAT.

Examples

vspatial -i input_image.xv -w 5 -h 5 -m 0 -v 1 -e 1 -o output_image

the input image must of type BYTE; the selected width and height of the window is 5 x 5, which will result in a border width of two pixels in the output image. The mean spatial operator is unselected (default is selected), and the variance and entropy operations are selected, which will result in an output image (feature vector) of 2 bands, based on the operations selected.

Restrictions

All input images must be of data storage type BYTE.

All output images are of data storage type FLOAT.

Command Line Arguments

```
Usage for vspatial: Compute Spatial Features Using NxM Window  (K1)
% vspatial
  -i         (infile)  input image filename
  -o         (outfile) output image filename

  [-w]       (integer) window width argument
             (1 <= w <= 64, default = 3)
  [-h]       (integer) window height argument
             (1 <= h <= 64, default = 3)
  [-m]       (boolean) apply the Mean Spatial Operator
             (default = true)
  [-s]       (boolean) apply the Angular 2nd Moment Spatial Operator
             (default = false)
  [-c]       (boolean) apply the Contrast Spatial Operator
             (default = false)
  [-v]       (boolean) apply the Variance Spatial Operator
             (default = false)
  [-e]       (boolean) apply the Entropy Spatial Operator
             (default = false)
  [-d]       (boolean) apply the Dispersion Spatial Operator
             (default = false)
```

B.47. vspeckle — *Reduce Speckle Noise Using the Crimmins Algorithm (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Spatial Filters*
Operator:	*Speckle Removal*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vspeckle reduces the "speckle" noise from an image. It uses the "eight hull algorithm" developed by T. R. Crimmins to perform this reduction.

The function takes one argument: *n* which determines how many times the image is to be sent through the speckle reducing filter. This filter uses the complementary hulling technique to reduce the speckle index of an image.

Each pixel in the image is compared with all eight of its surrounding pixels. The pixels above and below the pixel are the N-S pair, the pixels on either side are the E-W pair, those on the diagonals are the NW-SE and NE-SW pair. The image and a pair of pixels are sent to the symmetric hull function which in turn sends the image to the positive hull function and to the negative hull function. The positive and negative hull functions are called twice; the first time the neighboring pair of pixels is sent and the second time the complement of the pixels is sent. The positive hull function replaces each "middle pixel" with the result of a several comparisons. These comparison are a combination of maximums and minimums. The final comparison is a maximum for the positive hull algorithm. The image is then sent to the negative hull function where the middle pixel is replaced with the result of more comparisons. The final comparison for the negative hull algorithm is a minimum. The result of all these replacements has the effect of reducing the undesired speckle noise while preserving the edges of the original image. The Geometric filter for speckle reduction is described in *Applied Optics*, Vol. 24, No. 10, 15 May 1985, *Geometric fileter for speckle reduction*, by Thomas R Crimmins.

Examples

vspeckle -i image1 -o image2 -n 2

This command sends the image through the geometric filter twice. Most of the speckle noise is removed from image1 and the result is stored in image2.

See Also

lvspeckle(3)

Restrictions

vspeckle accepts only BYTE images. The number of iterations is limited to 20.

Command Line Arguments

```
Usage for vspeckle: Reduce Speckle Noise Using the Crimmins Algorithm  (K1)
% vspeckle
  -i          (infile)  input image
  -o          (outfile) output image

  [-n]        (integer) Iterations
              (1 <= n <= 20, default = 1)
```

B.48. vsurf — *Compute Surface Parameters (Normals and Angles) (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Surface*
Operator:	*Normals and Angles*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vsurf will take a surface image and produce two output images: one with the normals for each point on the surface and one with the angle between the surface normal and the normal to the zero plane of the image.

The angle output image is in degrees.

Examples

% vsurf -i surface -norm normals -ang angles

Restrictions

Works only on FLOAT input images.

Command Line Arguments

```
Usage for vsurf: Compute Surface Parameters (Normals and Angles)  (K1)
% vsurf
  -i          (infile)  input image
  -norm       (outfile) output normals image
  -ang        (outfile) output angles image
```

B.49. vtexture — *Compute Texture Features Using the LAW Metrics (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Feature Extraction*
Operator:	*Texture Extraction*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

NOTE: Program will NOT execute! vtexture functionality is replaced by DATAMANIP::ktexture. Command line arguments have changed slightly. The texture kernel switches (-L5L5, etc) have been changed from booleans to flags.

vtexture will extract texture features from an image by convolving the original image with a spatial filter. The spatial filters are comprised of 5 x 5 kernels derived from center-weighted vectors defined by the LAWS texture measure.

The five center-weighted vectors are:

L5 = [1 4 6 4 1]
E5 = [-1 -2 0 2 1]
S5 = [-1 0 2 0 -1]
W5 = [-1 2 0 -2 1]
R5 = [1 -4 6 -4 1]

Each 5 x 5 kernel is derived from multiplying a particular combination of two of the above vectors. This results in 25 possible 5 x 5 kernels. Note that 10 of the kernels are formed by taking the transpose. For example, L5E5 is a 5 x 5 kernel formed by multiplying the vectors, L5 with E5. It's transpose is E5L5, and is similarly formed by multiplying the vectors, E5 with L5. One of these kernels is more sensitive to horizontal changes in texture, while the other is sensitive to vertical changes in texture.

The sum of the elements of each kernel is zero, which results in the output image having a mean of zero. Therefore the relevant texture information is contained in the image variance of the microtexture features. The LAW kernels were designed to be sensitive to structures such as edges, ripples, and spots.

Choosing which kernel to use will depend on the nature of the texture of interest, and will require trial and error to decide which produces the feature with the most discriminating power. In general, if a texture kernel of say, L5S5 is used, one may also want to use its transpose, S5L5 for certain types of quasiperiodic variations commonly found in textured images.

Once the texture features are extracted, macrostatistical estimates may be obtained using a larger window size (eg. 15 x 15 or 31 x 31). Some useful statistics may be the variance or the absolute average. Such features of the texture filtered image were referred to as texture energy measures by K.I. LAWS.

See Also

DATAMANIP::ktexture RETRO::vspatial

Restrictions

will operate on images of data storage type BYTE or FLOAT. The output image will be either an INTEGER or FLOAT image, depending on the data type of the input image. If the input image is of data type BYTE or SHORT, then it is converted up to an INTEGER image. If the input image is of data type FLOAT, then the resulting output image will be of data type FLOAT.

References

vtexture K. I. Laws, "Rapid texture identification" Proc. SPIE, vol. 238, pp.376-380, 1980.

J. Y. Hsiao, and A. A. Sawchuk, "Supervised textured image segmentation using feature smoothing and probabilistic relaxation techniques" IEEE Trans. Pattern Anal. Machine Intell., vol. 11, No. 12, pp 1279-1292, 1989.

Command Line Arguments

```
Usage for vtexture: Compute Texture Features Using the LAW Metrics  (K1)
% vtexture
  -i          (infile)  input image filename
  -o          (outfile) output image filename

  [-L5L5]     (boolean) if true, apply L5L5 texture kernel
              (default = true)
  [-E5L5]     (boolean) if true, apply E5L5 texture kernel
              (default = false)
  [-S5L5]     (boolean) if true, apply S5L5 texture kernel
              (default = false)
  [-W5L5]     (boolean) if true, apply W5L5 texture kernel
              (default = false)
  [-R5L5]     (boolean) if true, apply R5L5 texture kernel
              (default = false)
  [-L5E5]     (boolean) if true, apply L5E5 texture kernel
              (default = false)
  [-E5E5]     (boolean) if true, apply E5E5 texture kernel
              (default = false)
  [-S5E5]     (boolean) if true, apply S5E5 texture kernel
              (default = false)
```

```
[-W5E5]    (boolean) if true, apply W5E5 texture kernel
           (default = false)
[-R5E5]    (boolean) if true, apply R5E5 texture kernel
           (default = false)
[-L5S5]    (boolean) if true, apply L5S5 texture kernel
           (default = false)
[-E5S5]    (boolean) if true, apply E5S5 texture kernel
           (default = false)
[-S5S5]    (boolean) if true, apply S5S5 texture kernel
           (default = false)
[-W5S5]    (boolean) if true, apply W5S5 texture kernel
           (default = false)
[-R5S5]    (boolean) if true, apply R5S5 texture kernel
           (default = false)
[-L5W5]    (boolean) if true, apply L5W5 texture kernel
           (default = false)
[-E5W5]    (boolean) if true, apply E5W5 texture kernel
           (default = false)
[-S5W5]    (boolean) if true, apply S5W5 texture kernel
           (default = false)
[-W5W5]    (boolean) if true, apply W5W5 texture kernel
           (default = false)
[-R5W5]    (boolean) if true, apply R5W5 texture kernel
           (default = false)
[-L5R5]    (boolean) if true, apply L5R5 texture kernel
           (default = false)
[-E5R5]    (boolean) if true, apply E5R5 texture kernel
           (default = false)
[-S5R5]    (boolean) if true, apply S5R5 texture kernel
           (default = false)
[-W5R5]    (boolean) if true, apply W5R5 texture kernel
           (default = false)
[-R5R5]    (boolean) if true, apply R5R5 texture kernel
           (default = false)
```

B.50. vtilt — *Add a Specified Illumination Gradient Plane to Image (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Surface*
Operator:	*Add Tilt*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

vtilt adds tilt to an image. Often an image will have a skew in the grey level values that represents a plane with a certain slope. The tilt takes the form of a plane which has an x and y zero crossing at the center of the image with x an y slope as specified in the argument list. The x and y slopes must be given in units per pixel and the x and y values default to 0.0.

The optional mask image (-m) must be of the same data type as the input images, and is used to gate the operation. A non-zero mask pixel enables the output pixel to contain the adjusted value. A zero mask pixel just transfers the value of the corresponding pixel in the first input image to the output pixel.

Examples

vtilt -i image -o out -x 2 -y -1

See Also

vgettilt, lvgettilt, vdetilt

Restrictions

vtilt will not operate on COMPLEX or BIT data storage types.

Command Line Arguments

```
Usage for vtilt: Add a Specified Illumination Gradient Plane to Image  (K1)
% vtilt
  -i         (infile)  input image
  -o         (outfile) resulting output image

  [-m]       (infile)  tilt operation masking image
             (default = {none})
  [-x]       (float)   specifies the slope in the X direction
             (unbounded, default = 0)
  [-y]       (float)   specifies the slope in the Y direction
             (unbounded, default = 0)
```

B.51. vwiener — *Image Restoration by Wiener (Least-Squares) Filtering (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Frequency Filters*
Operator:	*Wiener Restoration*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

NOTE: Program will NOT execute! vwiener functionality is replaced by DATAMANIP::kwiener. Command line arguments have changed slightly. -o1 is replaced by -o and -i3, -o2, and -t have gone away.

vwiener performs restoration of images in the spatial domain by Wiener filtering in the frequency domain. The idea is to deconvolve the blurring function from the blurred image. A few problems exist in performing this operation, as described in great detail in any good image processing book. The Wiener filter performs better than the inverse filter when noise is present in the data.

The major difficulty lies in how small the denominator is allowed to be before you call it garbage. The value chosen is supplied as the -t thresh value. Ideally, this will be just above the noise floor in the image, but experimentation is the best guide here.

The restoration parameter controls how smoothly the modified PSF blends into the flat cutoff regions generated by the thresholding operation. For more information, see Rosenfeld & Kak, page 287.

The reffile is an image which, if supplied, is subtracted from the input image before the wiener filter is applied. This is handy for removing some constant background from the image before the restoration is done.

The PSFfile is the transform of the Point-Spread Function, or blurring function that is to be deconvolved from the input image.

The effPSFfile is the same as the PSFfile after the thresholding operation has been applied. Taking the inverse Fourier transform of this file gives the *effective* PSF as used in the restoration. The idea is that is should be reasonably close to the real PSF.

Examples

vwiener -i1 fft_img1 -i2 fft_psf_img -t 10 -w 5 -o1 fft_out_img

This command deconvolves the second input image from the first. Fft_img1 is the FFT of the image to deconvolve from, and fft_psf_img is the FFT of the PSF function of the image to deconvolve. The threshold for the denominator is 10, and the resulting FFT image is fft_out_img. The Wiener parameter is 5.

See Also

DATAMANIP::kwiener

Restrictions

all input images must be the same data storage type (VFF_TYP_COMPLEX) and size.

Command Line Arguments

```
Usage for vwiener: Image Restoration by Wiener (Least-Squares) Filtering  (K1)
% vwiener
  -i1         (infile)  FFT of image to be restored
  -i2         (infile)  FFT of PSF function
  -o1         (outfile) FFT of restored image

  [-i3]       (infile)  FFT of reference image
              (default = {none})
  [-t]        (float)   minimum PSF threshold
              (t >= 0.0, default = 1e-06)
  [-w]        (float)   Wiener parameter
              (w >= 0.0, default = 1e-06)
  [-o2]       (outfile) FFT of restoration filter
              (default = {none})
```

B.52. vwmdd — *Weighted Minimum Distance Detector (K1)*

Object Information

Category:	*Khoros 1*	Subcategory:	*Classification*
Operator:	*Weighted Min Dist*	Object Type:	*kroutine*

In Cantata: *Yes*

Description

vwmdd is an Weighted minimum distance detector or Classifier. The main idea behind this detector is that it will distinguish a single class form the rest of the data. However, it will also work to distinguish multiple classes from each other.

To classify or detect spatial patterns in an image, we must have a template to compare against. Creating a template can be thought of as training on representative (ideal) data. This can be done by using a clustering algorithm on the representative data. Clustering routines such as vkmeans, vquant, isodata, etc can be used to create a template. The idea here is to over cluster the representative data, and quantize that into k classes. Pseudo color in editimage, and kasc2val can be used to create a class image. When the clustering is performed, two items need to be computed; (1) the cluster center values, and (2) the variances for each axis in the cluster. The variances can be obtained by looking at the diagonal of the covariance matrix.

Explained below is a method of creating a the class assignments for each cluster center. The class assignments may, for example, quantize the space from 12 clusters to 2 classes. Example

```
Cluster Center        Class
   1 ---------------------  2
   2 ---------------------  2
   3 ---------------------  1
   4 ---------------------  2
        .
        .
        .
  12 ---------------------  1
```

This routine expects an image that contains the diagonal terms of the covariance matrix, called the variance image, and an image that contains the cluster center values and class assignment for each cluster center. And of course the image to be classified. This detector is to approximate the following equation:

(X - Mi)˜ Inv(C) (X - Mi) (1)

where Mi is the mean, inv(C) is the inverse of the Covariance matrix, and X is the data points, and ˜ denotes transpose. This can be written as

```
(d1 d2 ... dn)˜ - Q1            - (d1 d2 ... dn)       (2)
                |    Q2         |
                |      .    0   |
                |        .      |
```

```
               |  0       .   |
               |          Qn  |
               -              -

    Which equals

        sq(d1)   sq(d2)           sq(dn)
        ------ + ------ + ..... + ------                (3)
          Q1       Q2               Qn

    Notation:
         sq => square
         Qi => the ith variance
         di => the ith (X - Mi) value
         the matrix is actually the inverse of C
```

Since the inverse of the Covariance matrix can not be easily determined, we only consider the diagonal terms or simply the variance terms.

There are two methods for detecting classes in the input image. Each data point is put through a test based on the ideas above. The detector works in two modes as follows:

(1) uses the summed method -s option set to yes. This is an approximation to method (2).

```
         sq( || X - Mi || )   <
    Vi = ------------------   >   1
             K x sq(Si)

    where:    sq( || X - Mi || ) =  sq(di)
                                 = Euclidean Distance Squared
              sq(Si) = trace(C)
                 the trace of C is the sum of the
                 diagonal terms (variances).
              K is a Constant
```

(2) non summed method (default).

```
          sq(d1)   sq(d2)           sq(dn)   <
    Vi =  ------ + ------ + ..... + ------   >   1
          Q1 x K   Q2 x K           Qn x K

    where:  sq(dj) = sq( || xj - Mi || )
                   = Euclidean Distance Squared of the
                     jth element of the vector X.
            Qj = the jth variance element of the variance
                 vector for the ith cluster center.
```

```
K = Constant
```

In both cases the constant K is used. This is used to adjust the variance value(s). If K is large, Qi can increase such that Vi is always less than 1. There are no specific rules as to the optimal value for K.

The data point X is put through the test for each cluster, the data point X is assigned to a Class based on the following criteria:

1. choose the smallest Vi value, where Vi is the result for cluster i.

2. If Vi is greater than 1, assign X to the NULL class (NULL class is always 0). Otherwise assign Vi to class that corresponds to the ith cluster.

This routine accepts three images as the input. The image that corresponds to the -i1 argument is the image that needs to be classified. This image can be have as many data bands as necessary. The number of data bands depicts the dimensionality of the vectors. Each pixel in the image is a vector with dimensionality of the number of data bands.

The second input image corresponds to the -i2 argument and is the prototype image. This image contains two pieces of information. (1) the cluster center values for each cluster center, and (2) the class assignments for each cluster center. The last data band of this image contains the class assignments for each cluster center value. This image must contain the same number of data bands as the other input image plus an extra data band that represents the class mapping. This image would most likely have been created by vkmeans or some other routine that will give cluster centers. This image contains vectors that correspond to the prototype of each class.

As stated above the center image's last data band is a class data band. The class data band simply maps each cluster center vector to a final class.

At this point most class images must be created manually. A class image can be created by using pseudo color in editimage on the resulting clustered image from the vkmeans routine. Vcustom can then be used to create the class image. Finally vinset can be used to add the class image data to the center image. Normally one would over cluster using vkmeans then reduce the space down. So vkmeans might produce 100 clusters but, really only 3 classes are desired. Pseudo color in editimage and kasc2val can be used to reduce the space down and create a class image. The class data might map cluster center vectors 1-50 to class 1, vectors 51-60 to class 2 and vectors 61-100 to class 3. When the vwmdd routine is run, the result would be a classified image of three classes.

The third input image corresponds to the -i3 argument. This image contains the variance values for each cluster center. This image contains the same number of data bands as the input image (-i1 image), and contains the same number of vectors as the center image (-i2 image). Each element in the variance vector is the variance value along an axis in the n-dimensional space. The variance values can be found by computing the diagonal of the covariance matrix for each cluster. The cluster center values would be used as the mean values. Note, vkmeans will compute this image for you.

The scale factor (-k) must be chosen by trial and error. It has been found that if the summed method is being used a small (1.0 - 3.0) value for k is sufficient. If the average method is not being used a larger k

(8.0 - 10.0) value is sufficient. Note, this may change based on the data, so these ranges are only a starting point.

The border option (-b) allows the user to specify a border width, in pixels, encompassing the image. The border region is skipped by vwmdd when classification is performed. This useful if neighborhood operators have been used previously and have not updated edge pixels.

All input images must be of data storage type FLOAT.

This routine was written with the help of and ideas from Dr. Don Hush, University of New Mexico, Dept. of EECE.

Examples

vwmdd -i1 aerial_image.xv -i2 centers_class -i3 variance -b 5 -k 8.0 -o test

aerial_image is a 5 band image; thus, each vector has dimensionality of 5, centers_class image is a 6 band image, 5 bands that contain the cluster center values and 1 class band. The border that vwmdd will ignore is 5 pixels wide. The output image will be a single band image called test. The scalar to multiply the denominator by is 8.0. By default the non-summing method is used.

See Also

vkmeans(1)

Restrictions

All input images must be of data storage type FLOAT.

Command Line Arguments

```
Usage for vwmdd: Weighted Minimum Distance Detector  (K1)
% vwmdd
  -i1        (infile)  Input Image
  -i2        (infile)  Input Cluster Center/Class Image
  -i3        (infile)  Input Variance Image
  -o         (outfile) Output specifying which vector belongs to which cluster

  [-k]       (float)    Scaling factor
             (k > 0.0, default = 1)
  [-b]       (integer)  Border width in pixels
             (0 <= b <= 100, default = 0)
  [-s]       (boolean) If true, Use Summed Method
             (default = false)
```

Table of Contents

A. Introduction 2-1

B. Khoros 1.0 Image Processing & Analysis Operators 2-1

B.1. lrfclass — *Classify Image Using the Localized Receptive Field Classifier (K1)* 2-1

B.2. lrftrain — *Calculate Weights for Localized Receptive Field Classifier (K1)* 2-4

B.3. v1bgamut — *Compress Grey Levels of Image (Produces Image with Map) (K1)* 2-8

B.4. vbandsp1 — *Extract a Specified Data Band from a Multiband Image (K1)* 2-9

B.5. vbandsp3 — *Extract 3 Bands from a Multiband Image (K1)* 2-10

B.6. vcast — *Upconvert All Inputs to the Highest Input Data Type (K1)* 2-12

B.7. vclose — *Close Boundaries of an Edge Image (K1)* 2-13

B.8. vcltrans — *Convert Between Tristimulus Color Spaces* 2-14

B.9. vcmtrans — *Linear Color Space Conversion with Specified 3x3 Matrix (K1)* 2-17

B.10. vconvert — *Convert Data Storage Type (K1)* 2-19

B.11. vconvolve — *Convolution of an Image and a Kernel (K1)* 2-20

B.12. vcost — *Compute Cost (Surface Arc Length) for Image (K1)* 2-21

B.13. vdetilt — *Remove Illumination Gradient by Subtracting Best-Fit Plane (K1)* 2-22

B.14. vdistance — *Compute Distance from the Boundary in a Two-Valued Image (K1)* 2-23

B.15. vdrf — *Optimal Difference Recursive Filter for Edge Detection (K1)* 2-24

B.16. vdyth — *Perform Dynamic Thresholding on Image (K1)* 2-31

B.17. venhance — *Enhance Image Using the Local Standard Deviation & Mean (K1)* 2-32

B.18. vextract — *Extract Rectangular Region from Object (K1)* 2-33

B.19. vfht — *Compute 2-D Fast Hadamard Transform (K1)* 2-34

B.20. vfractal — *Fractal Feature Extraction (K1)* 2-36

B.21. vgamth — *Generate a Binary Image by Thresholding a Color Image (K1)* 2-38

B.22. vgbox — *Create Image of Parallelogram Projected onto CCD Sensor (K1)* 2-39

B.23. vgcirc — *Create Image of a Filled Circle (K1)* 2-41

B.24. vgef — *First Derivative Operator for Symmetric Exponential Filter (K1)* 2-42

B.25. vgettilt — *Compute the Best-Fit Plane Parameters for Input File (K1)* 2-50

B.26. vgfractal — *Create Fractal Image with Specified Fractal Dimension (K1)* 2-51

B.27. vgpwl — *Create 2D Piecewise Linear Periodic Function (K1)* 2-54

B.28. vgrow — *Compute the Median Axis Inverse Transform (K1)* 2-55

B.29. vhmed — *Median Filter the Image Using Histogram to Find Median (K1)* 2-56
B.30. vhxray — *Stretch or Equalize Histogram Using Overlapping Windows (K1)* 2-57
B.31. vinverse — *Image Restoration by Inverse Filtering in Frequency Domain (K1)* 2-58
B.32. vkmeans — *Perform the K-means Clustering Algorithm on Image (K1)* 2-60
B.33. vlabel — *Perform Labeling on Multiband or Cluster Image (K1)* 2-63
B.34. vmarr — *Create a Marr Type Edge Detection Filter Kernel (K1)* 2-67
B.35. vmediaxis — *Compute the Medial Axis Transform (K1)* 2-67
B.36. vmindis — *Minimum Distance Classifier (K1)* 2-69
B.37. vmos2band — *Convert Mosaic Image to Multiband Image (K1)* 2-70
B.38. vpml — *Estimate Fractal Dimension of Image Based on P(m,L) (K1)* 2-73
B.39. vqerr — *Compute Error between Quantized Image and Original (K1)* 2-75
B.40. vqmed — *Median Filter the Image Using Quicksort to Find Median (K1)* 2-76
B.41. vquant — *Perform N-Dimensional Vector Quantization or Classification (K1)* 2-77
B.42. vsamp — *Sample an Image Along an Arbitrary Line (K1)* 2-78
B.43. vsdef — *Second Derivative Operator for Symmetric Exponential Filter (K1)* 2-80
B.44. vshape — *Perform Shape Analysis on Image (K1)* 2-88
B.45. vslope — *Compute Slope and Aspect Images from Elevation Data (K1)* 2-94
B.46. vspatial — *Compute Spatial Features Using NxM Window (K1)* 2-96
B.47. vspeckle — *Reduce Speckle Noise Using the Crimmins Algorithm (K1)* 2-98
B.48. vsurf — *Compute Surface Parameters (Normals and Angles) (K1)* 2-99
B.49. vtexture — *Compute Texture Features Using the LAW Metrics (K1)* 2-100
B.50. vtilt — *Add a Specified Illumination Gradient Plane to Image (K1)* 2-102
B.51. vwiener — *Image Restoration by Wiener (Least-Squares) Filtering (K1)* 2-103
B.52. vwmdd — *Weighted Minimum Distance Detector (K1)* 2-104