Package ‘imagefx’

October 13, 2022

Type Package

Title Extract Features from Images

Version 0.4.1

Author Alex J.C. Witsil

Maintainer Alex J.C. Witsil <alexjcwitsil@gmail.com>

Description Synthesize images into characteristic features for time-series analysis or machine learning applications. The package was originally intended for monitoring volcanic eruptions in video data by highlighting and extracting regions above the vent associated with plume activity. However, the functions within are general and have wide applications for image processing, analyzing, filtering, and plotting.

Depends R (>= 3.5.0)

Imports moments, signal

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Suggests knitr, rmarkdown, jpeg

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2020-02-13 18:40:02 UTC

R topics documented:

`amp.sig` *Amplify Time Varying Signals in Video*

Description

Filters then amplifies each pixel's time series signal.

Usage

```
amp.sig(img.list, fps, f.corners, amp ,n.poles, type)
```

Arguments

`img.list`	A series of images saved in a list. Each list element is a matrix that represents pixel values of an image, the dimensions of which correspond to the rows and columns of the matrix.
`fps`	Sample rate of video in frames per second (FPS). Each list element of `img.list` should be separated by `1/fps` seconds. Defaults to 30.
`f.corners`	Corners to use in the filter. Currently only a Butterworth filter is implemented.
`amp`	Scalar to multiply filtered signals by. Defaults to 10.
`n.poles`	Number of poles used in Butterworth filter. Defaults to two.
`type`	Type of filter used in Butterworth filter (i.e. 'low', 'high', 'stop', 'pass'. Defaults to 'pass' if `n.poles==2` and 'high' if `n.poles==1`.

Details

This algorithm is based on the landmark study by Wu et. al (2012) though is organized and implemented slightly differently. Namely, this algorithm takes advantage of R functionality instead of the original development language, MATLAB.

If the goal is to amplify subtle time variations in an image (e.g. heartbeat, breathing, volcanic emissions, etc...) choose corner frequencies that contain the desired signal. For example, most adult resting heart rates range between 1 and 1.5 Hz. To amplify these signals, use a filter type of either `low`, `high`, or `pass`. If the goal is to dampen or remove signals, choose appropriate corner frequencies and use a filter type of `low`, `high`, or `stop`. To help choose appropriate corner frequencies, see `sig.extract`.

Filtering is accomplished via the `apply` function for speed. However, if the dimensions of the images are large (i.e. `dim(img.list[[1]])`) or if the number of frames is large (i.e. `length(img.list)`), the RAM may fill up and the R session may crash.

Value

List with the same length and element dimensions as `img.list`. Each list element contains the amplified version of the original image list.

Author(s)

Alex J.C. Witsil

References

Wu, Hao-Yu, et al. "Eulerian video magnification for revealing subtle changes in the world." (2012)

See Also

`sig.extract`

Examples

```
##############################
### SYNTHETIC VIDEO INPUTS ###
##############################

## x and y dimension of the frame
dim.x = 64
dim.y = 64

## sample rate in frames per second
fps = 30

## how many seconds does the video last
n.secs =3

## what is the frequency at which the boxed region oscillates
sig.f = 2
```

```
## what is the peak amplitude of the signal
sig.peak = 0.2

## size of the boxed region
box.width = 10
box.height = 10


##################################
### VIDEO AMPLIFICATION INPUTS ###
##################################

## how much to amplify the signal by
amp=500

## filter corners -- should contain the signal of interest
f.corners = c(sig.f-0.2, sig.f+0.2)

## number of poles in Butterworth filter
n.poles = 2

## type of filter
type = 'pass'


################################
### GENERATE SYNTHETIC VIDEO ###
################################

## use the inputs to generate an image list (i.e. video)
img.list <- gen.eg.img.list(dim.x, dim.y, fps, n.secs, sig.f, sig.peak, box.width, box.height)


################################
### AMPLIFY THE VIDEO SIGNAL ###
################################

## amplify the video signal
amp.list <- amp.sig(img.list, fps, f.corners, amp, n.poles, type)


################
### PLOTTING ###
################

## save the users original margins
mar.org  <- par()$mar

## generate a time axis
tax <- (1:(n.secs*fps)) / fps

## get the raw video into a spatiotemporal matrix
```

```
org.spat.temp <- matrix(unlist(lapply(img.list,rowSums)),ncol=length(tax))
amp.spat.temp <- matrix(unlist(lapply(amp.list,rowSums)),ncol=length(tax))

## define some example frames to plot
eg.frames = c(18,26,34,41)

## define a layout matrix
layout.mat <- matrix(c(1:length(eg.frames), rep(5,4),6:9,rep(10,4)),nrow=2,byrow=TRUE)
layout(layout.mat)

## plot some example 'frames' from the original video
i=1
while(i<=length(eg.frames)) {
    par(mar=c(2,1,3,1))

    ## make the current title
    c.main = paste('Org. Frame ', eg.frames[i], sep='')
    image2(img.list[[eg.frames[i]]],axes=FALSE,main=c.main,cex.main=2,asp=1)

    i=i+1
}

## plot the spatiotemporal variations of the original and amplified video
par(mar=c(2,1,3,1))
image2(org.spat.temp, ylab='y',axes=FALSE,xaxs='i',asp=1)
mtext('y', side=2,line=0.5)
axis(1)
box()

## plot the same example frames from the amplified video
i=1
while(i<=length(eg.frames)) {
    par(mar=c(2,1,3,1))
    ## make the current title
    c.main = paste('Amp. Frame ', eg.frames[i], sep='')

    ## add the image
    image2(amp.list[[eg.frames[i]]],axes=FALSE,main=c.main,cex.main=2,asp=1)

    i=i+1
}

par(mar=c(2,1,3,1))
image2(amp.spat.temp, xlab='',ylab='',axes=FALSE,xaxs='i',asp=1)
axis(3)
mtext('y', side=2,line=0.5)
mtext('frames', side=1,line=0.5)
box()


## set the layout and par back to the users original value
par(mfrow=c(1,1),mar=mar.org)
```

blob.extract *Extract Blob Region from Image (Matrix)*

Description

Find the predominant blob region in an image using a Laplacian of Gaussian (LoG) blob detection algorithm. Blob points are found using a connected component algorithm (see Details)

Usage

```
blob.extract(img, blob.point, win.size, gaus, lap)
```

Arguments

`img`	Matrix with numeric values representing pixel color intensities
`blob.point`	x,y locations of a point that is contained within the sought after blob region. Normally the image's max (or min) value location.
`win.size`	Window size used in connected component algorithm (see Details).
`gaus`	Low pass Gaussian mask that has same dimensions as img
`lap`	Optional high pass Laplacian filter of same dimensions as img. Defaults to standard 5-point stencil.

Details

The LoG algorithm first applies a Gaussian then a Laplacian operator to the image. The resulting image is binarized (values either 0 or 1) depending on sign of values in the LoG Image.

The blob x,y locations surrounding the `blob.point` and are found via a connected component algorithm. This algorithm is designed for speed and may miss x,y locations if the blob is highly irregular or concave. Adjusting the `win.size` can yield more accurate blob locations but has a slower run time.

Value

List of length 2 where

`xy.coords`	Matrix of x,y locations of blob in image
`bin.image`	Image (matrix) of same dimension of `img` that gives the binarized result from the LoG Blob Detection

Author(s)

Alex J.C. Witsil

See Also

`build.gaus` `filt3d`

Examples

```
############
### EG 1 ###
############
## example with synthetic data

## create an image that is a simple gaussian
img <- build.gaus(100,100,sig.x=2,sig.y=10,x.mid=80,y.mid=60)

## find the where the maximum value is
img.max <- which(img==max(img),arr.ind=TRUE)

## define a sigma for the low pass filter
sig=5

## define the low pass filter as another gaussian
lp.filt <- build.gaus(nrow(img),ncol(img),sig.x=sig)

## define a window size for the connected component algorithm
win.size=0.05

## perform the blob detection
blob <- blob.extract(img=img, blob.point=img.max,win.size=win.size,gaus=lp.filt)

####################
### PLOTTING EG1 ###
####################

## define x and y grid lines
grid.x <- 1:nrow(img)
grid.y <- 1:ncol(img)

dev.new()
close.screen(all.screens=TRUE)
split.screen(c(1,3))

screen(1)
image(grid.x,grid.y,img,main='Image')

screen(2)
image(grid.x,grid.y,blob$bin.image,col=gray.colors(2),main='Binarized LoG Image')

screen(3)
image(grid.x,grid.y,img,main='Img with Blob Detected')
points(blob$xy.coords,col='black',pch=16,cex=1)

## close the screens
close.screen(all.screens=TRUE)


############
### EG 2 ###
```

```
############
## example with volcano image data.
## This RBG image shows ash erupting above the crater, which is cropped out

data(sakurajima)

## crop accroding to these corner values
xleft = 1
xright = 188
ybottom = 1
ytop = 396

## crop the image using crop.image
cropped <- crop.image(sakurajima, xleft, ybottom, xright, ytop)

## redefine the crop image
img <- cropped$img.crop

######################
### PRE PROCESSING ###
######################

## separate the image into red, green, and blue images
r.img <- img[,,1]
g.img <- img[,,2]
b.img <- img[,,3]

## remove the mean
r.img <- r.img-mean(r.img)
g.img <- g.img-mean(g.img)
b.img <- b.img-mean(b.img)

## calculate the the plane trend...
r.img.trend <- fit3d(r.img)
g.img.trend <- fit3d(g.img)
b.img.trend <- fit3d(b.img)

## remove the trend
r.img.dtrend <- r.img-r.img.trend
g.img.dtrend <- g.img-g.img.trend
b.img.dtrend <- b.img-b.img.trend


################################
### SET UP SOME FILTER MASKS ###
################################

## define a sigma for the LP Gaussian Filter
gaus.sig=15

## build the Gaussian filter
gaus <- build.gaus(nrow(img),ncol(img),gaus.sig)
```

```
## find the extreme (absolute valued maximum) value of each RGB channel
blob.r.point <- which(abs(r.img.dtrend)==max(abs(r.img.dtrend)),arr.ind=TRUE)
blob.g.point <- which(abs(g.img.dtrend)==max(abs(g.img.dtrend)),arr.ind=TRUE)
blob.b.point <- which(abs(b.img.dtrend)==max(abs(b.img.dtrend)),arr.ind=TRUE)

## set a window size to be used in the connected component algorithm
win.size = 0.05

## extract the blob xy locations
blob.r <- blob.extract(r.img.dtrend,blob.r.point,win.size,gaus)
blob.g <- blob.extract(g.img.dtrend,blob.r.point,win.size,gaus)
blob.b <- blob.extract(b.img.dtrend,blob.r.point,win.size,gaus)


####################
### PLOTTING EG2 ###
####################

## note the blob points (blob$xy.coords) must be adjusted according to
## where the origin (0,0) is located in R plots image plots
blob.coords.r  <- blob.r$xy.coords
blob.coords.r[,1] <- blob.r$xy.coords[,2]
blob.coords.r[,2] <- (blob.r$xy.coords[,1]-nrow(r.img))*-1

blob.coords.g  <- blob.g$xy.coords
blob.coords.g[,1] <- blob.g$xy.coords[,2]
blob.coords.g[,2] <- (blob.g$xy.coords[,1]-nrow(g.img))*-1

blob.coords.b  <- blob.b$xy.coords
blob.coords.b[,1] <- blob.b$xy.coords[,2]
blob.coords.b[,2] <- (blob.b$xy.coords[,1]-nrow(b.img))*-1


## save the users options
mar.usr=par()$mar

dev.new()
close.screen(all.screen=TRUE)
par(mar=c(0,0,2,0))
split.screen(c(1,2))
split.screen(c(3,1),screen=2)

screen(1)
image2(sakurajima,asp=1,axes=FALSE)
rect(ybottom,nrow(sakurajima)-xleft,ytop,nrow(sakurajima)-xright,lwd=3,border='white',lty=3)
title('Original Image',line=0,font=2,col='white',cex=2,)

screen(3)
image2(r.img,asp=1,axes=FALSE)
points(blob.coords.r,col=rgb(1,0,0,alpha=0.05),pch=16,cex=0.3)
title('Red Channel',line=0,font=2,col='red',cex=2)

screen(4)
```

```
image2(g.img,asp=1,axes=FALSE)
points(blob.coords.g,col=rgb(0,1,0,alpha=0.05),pch=16,cex=0.3)
title('Green Channel',line=0,font=2,col='darkgreen',cex=2)

screen(5)
image2(b.img,asp=1,axes=FALSE)
points(blob.coords.b,col=rgb(0,0,1,alpha=0.05),pch=16,cex=0.3)
title('Blue Channel',line=0,font=2,col='darkblue',cex=2)

## return the users original margins and close screens
par(mar=mar.usr)
close.screen(all.screens=TRUE)
```

blob.stats *Blob Statistics from Erebus Volcano, Antarctica*

Description

Raw statistics output from `calc.blob.stats` that show a single bubble bursting event from the lava lake at Mount Erebus. Each list component corresponds to statistics calculated from the red, green, and blue color channels from the image frames.

Usage

```
data("blob.stats")
```

Format

The format is: List of 3 $ r: num [1:200, 1:14] 331 332 330 334 330 ... $ g: num [1:200, 1:14] 567.15 568.84 -3.13 -3.08 -4.33 ... $ b: num [1:200, 1:14] 454.8 473.3 461.6 476.2 -2.5 ...

Examples

```
data(blob.stats)
## maybe str(blob.stats) ; plot(blob.stats) ...
```

build.gaus *Build 2D Gaussian Image (Matrix)*

Description

Build a 2-dimensional Gaussian matrix for filtering, correlations, data testing, or other various uses.

Usage

```
build.gaus(xdim, ydim, sig.x, sig.y, x.mid, y.mid)
```

Arguments

`xdim`	size in the x dimension
`ydim`	size in the y dimension
`sig.x`	Gaussian sqrt(variance) in x direction
`sig.y`	Gaussian sqrt(variance) in y direction. Defaults to `sig.x` if undefined
`x.mid`	peak location in x direction
`y.mid`	peak location in the y direction

Details

Note that if `xdim` or `ydim` are even and `x.mid` and `y.mid` are left undefined, the Gaussian peak will be off center. This can be a problem when using a Gaussian matrix for multiple filtering operations.

Value

matrix with values corresponding to values in the 2D Gaussian.

Author(s)

Alex J.C. Witsil

Examples

```
############
### EG 1 ###
############

## define the dimensions of the gaussian image (matrix)
xdim=101
ydim=101

## sigma in the x direction.  The y sigma defaults to the sig.x if not supplied
sig.x=5

## build the first example
gaus1 <- build.gaus(xdim,ydim,sig.x)


###################
## PLOTTING EG 1 ##
###################
image(1:nrow(gaus1),1:ncol(gaus1),useRaster=TRUE,gaus1)


############
### EG 2 ###
############

## define the dimensions of the gaussian image (matrix)
```

```
xdim=101
ydim=201

## define a sigma in the both the x and y direction
sig.y=5
sig.y=20

## define the center (peak) location of the guassian
x.mid = 30
y.mid = 120

## now build the gaussian
gaus2 <- build.gaus(xdim,ydim,sig.x,sig.y,x.mid,y.mid)

##################
## PLOTTING EG2 ##
##################
image(1:nrow(gaus2),1:ncol(gaus2),useRaster=TRUE,gaus2)
```

---

build.lap *Build 5-Point Laplacian Stencil*

Description

Given an x and y dimension, build a five point stencil of matching dimensions whose values are either -4, 0, and 1 and whose sum = 0.

Usage

```
build.lap(xdim,ydim)
```

Arguments

`xdim`	x dimension of desired matrix output
`ydim`	y dimension of desired matrix output

Details

Note that if `xdim` or `ydim` are even, the stencil values will be off center. This can be a problem when using a Laplacian matrix for multiple filtering operations.

Value

Matrix with dimensions equal to `xdim` and `ydim` with five point stencil located near the middle of the matrix (see Details).

Author(s)

Alex J.C. Witsil

See Also

`build.gaus`

Examples

```
## build a 5 point stencil laplacian
lap=build.lap(9,9)
image(lap)
```

`calc.blob.stats` *Calculate Color and Spatial Statistics from Blob Region*

Description

Wrapper function (see details) that calculates various statistics on x,y data that corresponds to an image.

Usage

```
calc.blob.stats(img, xy.coords)
```

Arguments

`img`	Matrix whose values at `xy.coords` are statistically analyzed.
`xy.coords`	Index locations corresponding region of interest (e.g. blob region) in the `img`

Details

Function calls multiple statistical functions (e.g. `mean`, `sd`) and applies them to regions in the `img` according to the index locations given by `xy.coords`. In general, this function is commented to promote any modifications needed to fit the users needs. For example, adding or removing statistical analyses is straight forward.

Value

Numeric vector giving the statistics of the blob region.

Author(s)

Alex J.C. Witsil

See Also

`mean` `sd` `sum` `colMeans` `rowMeans` `length` `skewness` `kurtosis`

Examples

```
############
### EG 1 ###
############
## example with synthetic data

## create an image that is a simple gaussian
img <- build.gaus(100,100,sig.x=2,sig.y=10,x.mid=80,y.mid=60)

## find the where the maximum value is
img.max <- which(img==max(img),arr.ind=TRUE)

## define a sigma for the low pass filter
sig=5

## define the low pass filter as another gaussian
lp.filt <- build.gaus(nrow(img),ncol(img),sig.x=sig)

## define a window size for the connected component algorithm
win.size=0.05

## perform the blob detection
blob <- blob.extract(img=img, blob.point=img.max,win.size=win.size,gaus=lp.filt)

#################################
### CALCULATE BLOB STATISTICS ###
#################################

blob.stats <- calc.blob.stats(img, blob$xy.coords)
print(blob.stats)


############
### EG 2 ###
############

## example with volcano image data.
data(sakurajima)

######################
### PRE PROCESSING ###
######################

## crop accroding to these corner values
xleft = 1
xright = 188
ybottom = 1
ytop = 396

## crop the image using crop.image
cropped <- crop.image(sakurajima, xleft, ybottom, xright, ytop)
```

```
## redefine the crop image
img <- cropped$img.crop

## separate the image into red, green, and blue images
r.img <- img[,,1]
g.img <- img[,,2]
b.img <- img[,,3]

## remove the mean
r.img <- r.img-mean(r.img)
g.img <- g.img-mean(g.img)
b.img <- b.img-mean(b.img)

## calculate the the plane trend...
r.img.trend <- fit3d(r.img)
g.img.trend <- fit3d(g.img)
b.img.trend <- fit3d(b.img)

## remove the trend
r.img.dtrend <- r.img-r.img.trend
g.img.dtrend <- g.img-g.img.trend
b.img.dtrend <- b.img-b.img.trend


################################
### SET UP SOME FILTER MASKS ###
################################

## define a sigma for the LP Gaussian Filter
gaus.sig=30

## build the Gaussian filter
gaus <- build.gaus(nrow(img),ncol(img),gaus.sig)

## find the maximum value of each RGB channel
blob.r.point <- which(r.img.dtrend==max(r.img.dtrend),arr.ind=TRUE)
blob.g.point <- which(g.img.dtrend==max(g.img.dtrend),arr.ind=TRUE)
blob.b.point <- which(b.img.dtrend==max(b.img.dtrend),arr.ind=TRUE)

## set a window size to be used in the connected component algorithm
win.size = 0.05

## extract the blob xy locations
blob.r <- blob.extract(r.img.dtrend,blob.r.point,win.size,gaus)
blob.g <- blob.extract(g.img.dtrend,blob.r.point,win.size,gaus)
blob.b <- blob.extract(b.img.dtrend,blob.r.point,win.size,gaus)


#################################
### CALCULATE BLOB STATISTICS ###
#################################
```

```
r.blob.stats <- calc.blob.stats(r.img.dtrend, blob.r$xy.coords)
g.blob.stats <- calc.blob.stats(g.img.dtrend, blob.g$xy.coords)
b.blob.stats <- calc.blob.stats(b.img.dtrend, blob.b$xy.coords)

print(r.blob.stats)
print(g.blob.stats)
print(b.blob.stats)
```

cor.mat *Correlate Matrix Rows*

Description

Wrapper function of ccf used to find best lag time between two vectors.

Usage

```
cor.mat(x, y, ...)
```

Arguments

x	Vector.
y	Vector.
...	Additional arguments to pass to ccf (e.g. max.lag)

Value

Scalar indicating the lag associated with the maximum correlation value between x and y.

Author(s)

Alex J.C. Witsil

See Also

ccf

Examples

```
## generate a time axis
tax = seq(0,10,by=0.1)

## generate two signals with a phase offset
sig1 <- sin(2*pi*1/2*tax)
sig2 <- sin(2*pi*1/2*tax + pi/2)

best.lag <- cor.mat(sig1,sig2)
```

```
################
### PLOTTING ###
################

plot(sig1,type='l',col='blue',main=paste('lag is: ',best.lag,sep=''))
lines(sig2,col='green')
```

`crop.image` *Crop an Image*

Description

Crop an image (matrix or array) with either pre determined bottom-left and top-right locations or interactively.

Usage

```
crop.image(img, xleft, ybottom, xright, ytop, pick)
```

Arguments

`img`	matrix or array of image to crop
`xleft`	left extreme of crop area
`ybottom`	bottom extreme of crop area
`xright`	right extreme of crop area
`ytop`	top extreme of crop area
`pick`	logical value indicating whether crop region should be selected interactively (see details)

Details

if any of the `xleft`, `xright`, `ybottom`, `ytop` are missing, or if `pick` is `TRUE`, an interactive plot will aid in picking crop regions. The original image will be plotted and the user must first select the bottom left corner, then the top right corner of the desired crop area. A corresponding rectangle will be plotted indicating the current crop region. If the region is sufficient, the user should then click *crop* in the top right corner of the plotting area. If the region should be modified, the user should click *repick*.

Note that the `xleft`, `xright`, `ybottom`, and `ytop` locations correspond to R's reference frame for matrices, which can be confusing.

Value

List of length two with

`img.crop`	an object giving the cropped image with the same class (either matrix or array) of `img`
`img.corners`	a vector with length 4 giving the the left, right, bottom, and top crop coordinates in the original image.

Author(s)

Alex J.C. Witsil

See Also

locator

Examples

```
############
### EG 1 ###
############
## example where you know where to crop the image

sakurajima.crop <- crop.image(sakurajima,xleft=146,ybottom=7,xright=203,ytop=256)
split.screen(c(1,2))
screen(1)
image2(sakurajima,asp=1,main='Original')
screen(2)
image2(sakurajima.crop[[1]],asp=1,main='Cropped')

## close screens
close.screen(all.screens=TRUE)

############
### EG 2 ###
############
## example where you choose where to crop using interactive plot

sakurajima.crop <- crop.image(sakurajima)

split.screen(c(1,2))
screen(1)
image2(sakurajima,asp=1,main='Original')
screen(2)
image2(sakurajima.crop[[1]],asp=1,main='Cropped')
print(sakurajima.crop[[2]])

## close screens
close.screen(all.screens=TRUE)
```

`erebus` *Image of Erebus Volcano, Antarctica*

Description

JPEG image read in as an array where the third dimension corresponds to the red, green, and blue color channels. Note the image has been compressed significantly to reduce the memory size.

Usage

```
data("erebus")
```

Format

The format is: num [1:300, 1:400, 1:3] 0.019768 0.0011 0.002516 0 0.000495 ...

References

Witsil and Johnson (2018) <10.1016/j.jvolgeores.2018.05.002>

Examples

```
data(erebus)
image2(erebus,asp=1)
## maybe str(erebus) ; plot(erebus) ...
```

`erebus.40` *Image of Erebus Volcano, Antarctica*

Description

The 40th frame in a series of images that was recorded seconds prior to a bubble bursting event at Mount Erebus.

JPEG image read in as an array where the third dimension corresponds to the red, green, and blue color channels. Note the image has been compressed significantly to reduce the memory size.

Usage

```
data("erebus.40")
```

Format

The format is: num [1:225, 1:300, 1:3] 0.0588 0 0 0 0 ...

References

Witsil and Johnson (2018) <10.1016/j.jvolgeores.2018.05.002>

Examples

```
data(erebus.40)
image2(erebus.40)
```

erebus.70 *Image of Erebus Volcano, Antarctica*

Description

The 70th frame in a series of images that was recorded during a bubble bursting event at Mount Erebus.

JPEG image read in as an array where the third dimension corresponds to the red, green, and blue color channels. Note the image has been compressed significantly to reduce the memory size.

Usage

```
data("erebus.70")
```

Format

The format is: num [1:225, 1:300, 1:3] 0.06667 0 0 0.02353 0.00392 ...

References

Witsil and Johnson (2018) <10.1016/j.jvolgeores.2018.05.002>

Examples

```
data(erebus.70)
image2(erebus.70)
```

erebus.90 *Image of Erebus Volcano, Antarctica*

Description

The 90th frame in a series of images that was recorded seconds after a bubble bursting event at Mount Erebus.

JPEG image read in as an array where the third dimension corresponds to the red, green, and blue color channels. Note the image has been compressed significantly to reduce the memory size.

Usage

```
data("erebus.90")
```

Format

The format is: num [1:225, 1:300, 1:3] 0.03922 0.00392 0.01176 0 0.01569 ...

References

Witsil and Johnson (2018) <10.1016/j.jvolgeores.2018.05.002>

Examples

```
data(erebus.90)
image2(erebus.90)
```

`fftshift` *Shift Zero Frequency to Center*

Description

Rearranges matrix such that the first quadrant is swapped with the third and the second quadrant is swapped with the fourth.

Usage

```
fftshift(x)
```

Arguments

`x`	matrix whose quadrants should be shifted.

Details

This function is generally used after applying an `fft` to force the zero-frequency to the middle of the matrix.

Note that if the matrix `x` has even dimensions, the zero frequency will be 1 unit from the center.

Value

Shifted matrix with same dimensions as `x`

Author(s)

Alex J.C. Witsil

See Also

`fft`

Examples

```
## build the four components of a matrix with four values (i.e. 1:4)
x1 <- matrix(1,nrow=1,ncol=1)
x2 <- x1+1
x3 <- x2+1
x4 <- x3+1

## combine all components together
x <- rbind(cbind(x1,x2),cbind(x3,x4))

## shift the matrix
x.shift <- fftshift(x)

## note the difference of the shifted and original
print(x)
print(x.shift)

################
### PLOTTING ###
################
## note the difference of the shifted and original graphically

close.screen(all.screens=TRUE)
split.screen(c(1,2))
screen(1)
image(x,main='Original',col=rainbow(4))
screen(2)
image(x.shift,main='FFT Shifted', col=rainbow(4))

## close screens
close.screen(all.screens=TRUE)
```

`filt3d` *Filter Image (Matrix) with Mask via Convolution*

Description

Apply a filter mask to smooth, sharpen, shift, or otherwise modify an image (matrix)

Usage

```
filt3d(x, mask)
```

Arguments

`x`	Image (matrix) to be filtered
`mask`	Filter mask (matrix) with same dimensions as `x`

Details

x and mask are convolved in the frequency domain via multiplication and returned to the spatial domain using the fast Fourier transform.

Value

Filtered matrix with same dimensions as x.

Author(s)

Alex J.C. Witsil

See Also

fft fftshift

Examples

```
##########
## EG 1 ##
##########
## example of a low pass filter

## generate test data
data <- matrix(0,nrow=256,ncol=256)

box.width = 100
box.height = 100
box.mid=c(nrow(data)/2,ncol(data)/2)

## define where the box indices are
box.row.inds <- (box.mid[1]-box.width/2):(box.mid[1]+box.width/2)
box.col.inds <- (box.mid[2]-box.height/2):(box.mid[2]+box.height/2)

## create the box in the data matrix
data[box.row.inds,box.col.inds] = 1

## define the sigma in the low pass Gaussian filter
sig=5

## create a low pass Gaussian filter
gaus <- build.gaus(nrow(data),ncol(data),sig)

## filter the data matrix with the Gaussian filter mask
data.lp <- filt3d(data,gaus)


## PLOTTING EG1 ##

dev.new()
close.screen(all.screens=TRUE)
split.screen(c(1,3))
```

```
## set up some grid lines
grid.xs <- 1:nrow(data)
grid.ys <- 1:ncol(data)

screen(1)
image(grid.xs,grid.ys,data,col=gray.colors(200),useRaster=TRUE,main="Data")

screen(2)
image(grid.xs,grid.ys,gaus,col=gray.colors(200),useRaster=TRUE,main="Low Pass Gaussian Mask")

screen(3)
image(grid.xs,grid.ys,data.lp,col=gray.colors(200),useRaster=TRUE,main='Filtered Data')

## close screens
close.screen(all.screens=TRUE)


##########
## EG 2 ##
##########
## example of a high pass filter

## generate test data
data <- matrix(0,nrow=256,ncol=256)

box.width = 100
box.height = 100
box.mid=c(nrow(data)/2,ncol(data)/2)

## define where the box indices are
box.row.inds <- (box.mid[1]-box.width/2):(box.mid[1]+box.width/2)
box.col.inds <- (box.mid[2]-box.height/2):(box.mid[2]+box.height/2)

## create the box in the data matrix
data[box.row.inds,box.col.inds] = 1

## find the middle of the data matrix
mid <- c(nrow(data)/2,ncol(data)/2)

## create a 5-point Laplacian high pass filter
lap <- matrix(0,nrow=nrow(data),ncol=ncol(data))
lap[(mid[1]-1):(mid[1]+1),mid[2]] = c(1,-4,1)
lap[mid[1],(mid[2]-1):(mid[2]+1)] = c(1,-4,1)

## perform  high pass filter on the data using the Laplacian mask
data.hp <- filt3d(data,lap)


## PLOTTING EG2 ##

## set up some grid lines
grid.xs <- 1:nrow(data)
grid.ys <- 1:ncol(data)

dev.new()
close.screen(all.screens=TRUE)
split.screen(c(1,3))

screen(1)
image(grid.xs,grid.ys,data,col=gray.colors(200),useRaster=TRUE,main="Data")

screen(2)
image(grid.xs,grid.ys,lap,col=gray.colors(200),useRaster=TRUE,main="High Pass Laplacian Mask")

screen(3)
image(grid.xs,grid.ys,data.hp,col=gray.colors(200),useRaster=TRUE,main='Filtered Data')

## close screens
close.screen(all.screens=TRUE)


##########
## EG 3 ##
##########
## example of a shift transform filter

## generate test data
data <- matrix(0,nrow=256,ncol=256)

box.width = 100
box.height = 100
box.mid=c(nrow(data)/2,ncol(data)/2)

## define where the box indices are
box.row.inds <- (box.mid[1]-box.width/2):(box.mid[1]+box.width/2)
box.col.inds <- (box.mid[2]-box.height/2):(box.mid[2]+box.height/2)

## create the box in the data matrix
data[box.row.inds,box.col.inds] = 1

## create a delta function at some (x,y) location
delta.x = 80
delta.y = 180
delta <- matrix(0,nrow=nrow(data),ncol=ncol(data))
delta[delta.x,delta.y] = 1

## perform the shift transform filter
data.shift <- filt3d(data,delta)


## PLOTTING EG3 ##

## set up some grid lines
grid.xs <- 1:nrow(data)
grid.ys <- 1:ncol(data)
```

```
dev.new()
close.screen(all.screens=TRUE)
split.screen(c(1,3))

screen(1)
image(grid.xs,grid.ys,data,col=gray.colors(200),useRaster=TRUE,main="Data")

screen(2)
image(grid.xs,grid.ys,delta,col=gray.colors(200),useRaster=TRUE,main="Shift Delta Mask")

screen(3)
image(grid.xs,grid.ys,data.shift,col=gray.colors(200),useRaster=TRUE,main='Filtered Data')

## close screens
close.screen(all.screens=TRUE)
```

`fit3d` *Fit a Plane to Image (Matrix) with SVD*

Description

Find plane that best fits trend in image (matrix)

Usage

```
fit3d(mat)
```

Arguments

`mat`	image (matrix) of values to fit plane to

Details

This function returns the best fit plane with the DC offset included. In other words average of the matrix values is added to the best fit plane within the function.

Value

matrix with same dimensions as `mat` whose values represent the best fit plane.

Author(s)

Alex J.C. Witsil

See Also

`svd`

Examples

```
## break the RGB image into 3 matrices
img.r <- erebus[,,1]
img.g <- erebus[,,2]
img.b <- erebus[,,3]

## find the planar trend in the red channel
trend.r <- fit3d(img.r)
trend.g <- fit3d(img.g)
trend.b <- fit3d(img.b)

## subtract the red channel trend from the original red channel image
img.r.detrend <- img.r-trend.r
img.g.detrend <- img.g-trend.g
img.b.detrend <- img.b-trend.b

## combine the RGB detrended matrices into an array
img.detrend = array(dim=dim(erebus))
img.detrend[,,1] <- img.r.detrend
img.detrend[,,2] <- img.g.detrend
img.detrend[,,3] <- img.b.detrend

################
### PLOTTING ###
################

close.screen(all.screens=TRUE)
split.screen(c(1,3))
screen(1)
image2(erebus,asp=1,main='Original Image',ylab='image rows',xlab='image cols')
screen(2)
image2(trend.r,asp=1,main='Fitted Trend',ylab='image rows',xlab='image cols')
screen(3)
image2(img.detrend,asp=1,main='Detrended Image',ylab='image rows',xlab='image cols')

## close screens
close.screen(all.screens=TRUE)
```

`gen.eg.img.list` *Generate Example Image List Data*

Description

Generates synthetic video data comprising noise around each frame's peripheral and a boxed region whose pixel values oscillate according to the input parameters.

Usage

```
gen.eg.img.list(dim.x, dim.y, fps, n.secs, sig.f, sig.peak, box.width, box.height)
```

Arguments

dim.x	x dimension of the video frames. Defaults to 64 pixels.
dim.y	y dimension of the video frames. Defaults to 64 pixels.
fps	Sampling rate of the synthetic video in frames per second. Defaults to 30 frames per second.
n.secs	Number of seconds in the synthetic video. Defaults to 3 s.
sig.f	Frequency at which the boxed region's pixel values oscillate. Defaults to 2 Hz.
sig.peak	Peak of signal. Defaults to 0.2.
box.width	Width of box region. Defaults to 10 pixels.
box.height	Height of box region. Defaults to 10 pixels.

Details

Use this function to create synthetic video data in the list structure required for functions like `amp.sig` and `sig.extract`.

Note that noise in the frames fluctuates between -1 and 1. Therefore, if you choose `sig.peak<1` in each frame the boxed region's pixel values will be below the signal to noise ratio. If you were to 'play' this video (i.e. cycle through the list elements) the boxed region would be difficult or impossible to distinguish without first amplifying the signal using `amp.sig`.

Value

List whose elements correspond to individual video frames, each of which are separated by `1/fps`.

Author(s)

Alex J.C. Witsil

See Also

`amp.sig` and `sig.extract`

Examples

```
###############
### INPUTS ####
###############

## x and y dimension of the frame
dim.x = 64
dim.y = 64

## sample rate in frames per second
fps = 30

## how many seconds does the video last
n.secs =3
```

```
## what is the frequency at which the boxed region oscillates
sig.f = 2

## what is the peak amplitude of the signal
sig.peak = 0.2

## size of the boxed region
box.width = 10
box.height = 10

################################
### GENERATE SYNTHETIC VIDEO ###
################################

## use the inputs to generate an image list (i.e. video)
img.list <- gen.eg.img.list(dim.x, dim.y, fps, n.secs, sig.f, sig.peak, box.width, box.height)

## or use the defaults in the function
img.list <- gen.eg.img.list()
```

`image2` *Plot Images (Matrices) with Intuitive Axes*

Description

Plot images (array or matrix) using the `rasterImage` function.

Usage

```
image2(img, ...)
```

Arguments

`img`	numeric matrix or array to plot
`...`	additional arguments to pass to the `plot` function

Details

Note the difference in image orientation between `image()` and `image2()`. These two plots will have the origin in different locations and can make things tricky when picking areas to crop, etc...

Value

Nothing. Returns a plotting window.

Author(s)

Alex J.C. Witsil

See Also

`rasterImage`

Examples

```
image2(sakurajima)
```

---

pcorr3d *Phase Correlation of Images*

Description

Input two images (matrices) and perform phase correlation by multiplication in the frequency domain. Return the maximum phase correlation value, its associated shift vector (x and y), and the phase correlation matrix.

Usage

```
pcorr3d(img1,img2)
```

Arguments

`img1`	Image (matrix) 1
`img2`	Image (matrix) 2 with same dimensions of `img1`

Details

Phase correlation calculated in the frequency domain as a multiplication. The dimensions of `img1` and `img2` must match. If `pcorr3d` is used to apply a match filter, it is logical to input the image to be searched over as `img1` and the match filter as `img2`. Similarly, if tracking relative motion between images, it is logical to input the first image at time t=n as `img1` and the second image at time t=n+1 as `img2`, otherwise motions will backwards.

Value

List whose values correspond to:

`max.shifts`	Vector of length two whose values are the x and y indices associated with the highest phase correlation value. Note these values are shifted according to the zero frequency which changes depending on the dimensions of `img1` and/or `img2`.
`max.corr`	Highest normalized phase correlation value in the correlation matrix
`corr.mat`	Normalized phase correlation matrix

Author(s)

Alex J.C. Witsil

See Also

xcorr3d

Examples

```
#################
### Example 1 ###
#################
## track movement between images

## load in the images
data(tux1,tux2)

## now find the shift vector by using the phase correlation function
shifts <- pcorr3d(tux1,tux2)

## ---- Plotting Example 1  ----- ##

split.screen(c(1,2))
screen(1)
image(1:nrow(tux1),1:ncol(tux1),tux1,col=gray.colors(200))

## define an example arrow starting and end points based on the shift found
x0 = nrow(tux1)/2
y0 = ncol(tux1)/2
x1 = x0 + shifts$max.shifts[1]
y1 = y0 + shifts$max.shifts[2]

## add arrows indicating how the image shifted
arrows(x0,y0,x1,y1)

## add a point where the arrow is
points(nrow(tux1)/2+shifts$max.shifts[1],ncol(tux1)/2+shifts$max.shifts[2],pch=21,bg='green')

screen(2)
image(1:nrow(tux2),1:ncol(tux2),tux2,col=gray.colors(200))
points(nrow(tux1)/2+shifts$max.shifts[1],ncol(tux1)/2+shifts$max.shifts[2],pch=21,bg='green')

## close the screen
close.screen(all.screens=TRUE)
```

---

points2 *Plot points on image*

Description

This is a plotting function that should be used with `image2`. This function is in line with `image2`, which locates the origin to the top-left corner as opposed to the bottom-left as `image` does.

Usage

```
points2(x,y,img,...)
```

Arguments

`x`	x location of points to plot
`y`	y location of points to plot
`img`	original image (matrix) which was plotted using `image2`. This is supplied simply for dimensional information.
`...`	additional arguments to be passed to `points`.

Value

Nothing. This is a plotting function.

Author(s)

Alex J.C. Witsil

See Also

`image2`

Examples

```
## build a test matrix
mat = matrix(0,nrow=20,ncol=20)
mat[1,1]=1

image2(mat,axes=FALSE,xlab='',ylab='')
points2(1,1,img=mat,col='red',pch=16)
```

`range01` *Scale Object Values Between Zero and One*

Description

Force the minimum value and maximum value of an object to 0 and 1, respectively.

Usage

```
range01(x)
```

Arguments

`x`	Matrix, vector, or other R object.

Value

An object with same dimensions as x and whose values range between zero and one.

Author(s)

Alex J.C. Witsil

Examples

```
## generate a signal to normalize
sig <- 10*sin(2*pi*5*seq(0,1,by=0.001))

## normalize between 0 and 1
sig01 <- range01(sig)

## check the ranges
range(sig)
##[1] -10 10
range(sig01)
##[1] 0 1
```

---

run.avg *Perform Running Average via Convolution*

Description

Smooth input data by convolution it with with a boxcar function of specified width. This is done in the frequency domain using multiplication.

Usage

```
run.avg(data, win)
```

Arguments

`data`	signal (vector) to convolve.
`win`	width of the boxcar in samples.

Details

Convolution occurs in the frequency domain via a multiplication.

Value

Smoothed vector with the same dimension as `data`

Note

This function uses `fft` which can take significantly long run times if the input signal is prime or is divisible by few integers.

Author(s)

Alex J.C. Witsil

See Also

`fft` `fftshift`

Examples

```
## make a delta 2D (time series) function
delta <- rep(0,101)
delta[floor(length(delta)/2)] = 1

## define a window length to average over
win = 20

## filter the delta function...this will result in a boxcar
box.car <- run.avg(delta,win)

## note sum(box.car) should equal the sum of the original signal...
## in this case sum(box.car)==1

##############
## PLOTTING ##
##############

plot(delta,type='h',lwd=2)
lines(box.car,col='blue',lwd=2,type='h')

legend('topright',col=c('black','blue'),legend=c('delta','running average'),lwd=2)
```

`sakurajima` *Image of Sakurajima Volcano, Japan*

Description

JPEG image read in as an array where the third dimension corresponds to the red, green, and blue color channels. Note the image has been compressed significantly to reduce the memory size.

Usage

```
data("sakurajima")
```

Format

The format is: num [1:300, 1:400, 1:3] 0.471 0.475 0.475 0.478 0.475 ...

Examples

```
data(sakurajima)
image2(sakurajima,asp=1)
```

`shift.vec` *Shift Vector*

Description

Shift a vector to the left or right by a certain amount (i.e. perform a simple phase shift).

Usage

```
shift.vec(shift, vec)
```

Arguments

`shift`	Amount of samples to shift vector in either the positive or negative direction.
`vec`	Vector to shift.

Details

The shift is accomplished by padding the head or tail of the vector with `NA` values.

Value

Shifted vector.

Author(s)

Alex J.C. Witsil

Examples

```

## generate a delta function
vec=rep(0,5)
vec[3] = 1

## shift vector by -2
new.vec = shift.vec(-2,vec)
```

`sig.extract` *Extract Time-Series Signal from Video Data*

Description

This function investigates time variations of pixel values in video frames via both a simple stack and also a cross correlation analysis. Both methods attempt to highlight signals present within video data. See Details.

Usage

```
sig.extract(img.list, fps, base.vec, ...)
```

Arguments

`img.list`	A series of images saved in a list. Each list element is a matrix that represents pixel values of an image, the dimensions of which correspond to the rows and columns of the matrix.
`fps`	Sample rate of video in frames per second (FPS). Each list element of `img.list` should be separated by `1/fps` seconds. Defaults to 30.
`base.vec`	Vector to correlate all pixel signals with. Defaults to `rowMeans(img.series)`.
`...`	Additional arguments passed to ccf (i.e., `lag.max`).

Details

The algorithm first synthesizes the video frames into a matrix of time series signals. In other words, a time-series is generated for each pixel that records the value of that pixel in each frame. These original pixel time-series are then simply stacked and then analyzed in terms of their frequency components. Additionally, each pixel time-series is cross correlated with the `base.vec` and then shifted according to the lag associated with the maximum correlation value. These shifted time-series are then stacked and analyzed in terms of their frequency components.

Note the function gives two basic outputs. The first is a simple stack (without applying any shift to the pixel time-series), while the second applies a shift then stack. Both outputs may prove useful and should be investigated.

Value

List of length four whose elements correspond to both original and shifted stacks. Note the nomenclature is that lower case objects correspond to time-series data while upper case corresponds to frequency domain data.

`org`	Matrix of stacked time-series from original pixel values. The first column corresponds to the time axis while second column is the stacked values.
`ORG`	Matrix of stacked frequency components from original pixel values. The first column corresponds to the frequency axis while second column is the stacked frequency amplitudes.

`shifted`	Matrix of stacked time-series from shifted pixel values. The first column corresponds to the time axis while second column is the stacked values.
`SHIFTED`	Matrix of stacked frequency components from shifted pixel values. The first column corresponds to the frequency axis while second column is the stacked frequency amplitudes.

Author(s)

Alex J.C. Witsil

See Also

`gen.eg.img.list`

Examples

```
##############################
### SYNTHETIC VIDEO INPUTS ###
##############################

## x and y dimension of the frame
dim.x = 64
dim.y = 64

## sample rate in frames per second
fps = 30

## how many seconds does the video last
n.secs = 3

## what is the frequency at which the boxed region oscillates
sig.f = 2

## what is the peak amplitude of the signal
sig.peak = 0.5

## size of the boxed region
box.width = 20
box.height = 20


################################
### GENERATE SYNTHETIC VIDEO ###
################################

## use the inputs to generate an image list (i.e. video)
img.list <- gen.eg.img.list(dim.x, dim.y, fps, n.secs, sig.f, sig.peak, box.width, box.height)


#################################
### EXTRACT SIGNAL FROM VIDEO ###
```

```
#################################

sig.x <- sig.extract(img.list,fps)

################
### PLOTTING ###
################

## set up a plot
split.screen(c(1,2))
screen(1)
plot(sig.x$org,col='blue',type='l',main='Time Domain')
lines(sig.x$shifted,col='red')

screen(2)
plot(sig.x$ORG,col='blue',type='l',xlim=c(0,5),main='Frequency Domain')
lines(sig.x$SHIFTED,col='red')

## close the screens
close.screen(all.screens=TRUE)
```

`tux1` *Image of Tux*

Description

Gray scaled image of Tux, the Linux Penguin. Here Tux is centered in the image. Originally this was a gray scaled jpeg image.

Usage

```
data("tux1")
```

Format

The format is: num [1:56, 1:66] 0.996 0.996 0.996 0.996 0.996 ...

Examples

```
data(tux1)
## maybe str(tux1) ; plot(tux1) ...
```

tux2 *Image of Tux*

Description

Gray scaled image of Tux, the Linux Penguin. Here Tux is shifted to the bottom right of the image. Originally this was a gray scaled jpeg image.

Usage

```
data("tux2")
```

Format

The format is: num [1:56, 1:66] 0.996 0.996 0.996 0.996 0.996 ...

Examples

```
data(tux2)
## maybe str(tux2) ; plot(tux2) ...
```

xcorr3d *Normalized Cross Correlation of Images*

Description

Input two images (matrices) and perform normalized cross correlation by multiplication in the frequency domain. Return the maximum normalized cross correlation value, its associated shift vector (x and y), and the correlation matrix.

Usage

```
xcorr3d(img1,img2)
```

Arguments

`img1`	Image (matrix) 1
`img2`	Image (matrix) 2 with same dimensions of `img1`

Details

Correlation calculated in the frequency domain as a multiplication. The dimensions of `img1` and `img2` must match. If `xcorr3d` is used to apply a match filter, it is logical to input the image to be searched over as `img1` and the match filter as `img2`. Similarly, if tracking relative motion between images, it is logical to input the first image at time t=n as `img1` and the second image at time t=n+1 as `img2`, otherwise motions will backwards.

Value

List whose values correspond to:

`max.shifts`	Vector of length two whose values are the x and y indices associated with the highest correlation value. Note these values are shifted according to the zero frequency which changes depending on the dimensions of `img1` and/or `img2`.
`max.corr`	Highest normalized correlation value in the correlation matrix
`corr.mat`	Normalized correlation matrix

Author(s)

Alex J.C. Witsil

See Also

`pcorr3d`

Examples

```
############################
### Optical Flow Example ###
############################
## track movement between images

## set up the image domain
xdim = 256
ydim = 256

## shift vectors accounting for movement between the two images
x.vec = 100
y.vec = 100

## declare the indices where the Gaussian peak will be in the first image
gaus1.x = 50
gaus1.y = 50

## shift the Gaussian according to the shift vector
gaus2.x <- gaus1.x + x.vec
gaus2.y <- gaus1.y + y.vec

## create the first synthetic image
img1 = build.gaus(xdim,ydim,sig.x=10,x.mid=gaus1.x,y.mid=gaus1.y)

## create the second synthetic image
img2 = build.gaus(xdim,ydim,sig.x=10,x.mid=gaus1.x+x.vec,y.mid=gaus1.y+y.vec)

## now find the shift vector by using the cross correlation function
shifts <- xcorr3d(img1,img2)

#############################
```

```
### Plotting Optical Flow ###
#############################

split.screen(c(1,2))
screen(1)
image(1:xdim,1:ydim,img1)

## add arrows indicating how the image shifted
arrows(gaus1.x,gaus1.y,gaus1.x+shifts$max.shifts[1],gaus1.y+shifts$max.shifts[2])

## add a point where the arrow is
points(gaus1.x+shifts$max.shifts[1],gaus1.y+shifts$max.shifts[2],pch=21,bg='green')

screen(2)
image(1:xdim,1:ydim,img2)

## add the point showing where the arrow is pointing
points(gaus1.x+shifts$max.shifts[1],gaus1.y+shifts$max.shifts[2],pch=21,bg='green')

## close the screen
close.screen(all.screens=TRUE)
```

Index