Package 'hyperbrick'

October 13, 2022

Type Package

Title Accessory Tools for Preprocessing Hyper-Spectral Images

Version 1.0

Date 2022-03-29

Description Read and execute preprocessing procedures on hyper-spectral images. These type of sensor data are usually recorded in a raw format. This package contains some easy-to-use functions to promptly build the image with some basic radiometric calibrations, setting up the geographic information. Geometric correction can be done with band-to-band registration (translation and rotation). Further functionalities allow to compute sliding windows statistics over the image.

License GPL (>= 2)

Author Anderson Rodrigo da Silva [aut, cre] (<https://orcid.org/0000-0003-2518-542X>)

Maintainer Anderson Rodrigo da Silva <anderson.agro@hotmail.com>

Encoding UTF-8

LazyLoad true

Depends raster

Imports rgdal, OpenImageR, caTools, dfoptim, pbapply

Suggests knitr, rmarkdown, spelling

URL https://github.com/arsilva87/hyperbrick

BugReports https://github.com/arsilva87/hyperbrick/issues

RoxygenNote 7.1.2

Language en-US

NeedsCompilation no

Repository CRAN

Date/Publication 2022-04-01 08:00:08 UTC

R topics documented:

`hyperbrick-package` *Accessory Tools for (Pre)Processing Hyperspectral Images*

Description

Read and (pre)process hyperspectral images. These type of sensor data is usually recorded in some raw format. This package contains some easy-to-use functions to promptly build the image with some basic radiometric calibrations and setting up the spatial information. Geometric correction can be done with band-to-band registration (translation and rotation). Further functionalities allows to compute sliding windows statistics over the image.

Details

Package:	hyperbrick
Type:	Package
Version:	0.1
Date:	2021-11-22
License:	GPL (>= 2)

Note

hyperbrick is an ongoing project. Any and all criticism, comments and suggestions are welcome. URL https://arsilva87.github.io/hyperbrick/

Author(s)

Anderson Rodrigo da Silva

Maintainer: Anderson Rodrigo da Silva <anderson.agro@hotmail.com>

affineBrick *Affine Transformation (rotation and shift) on Images*

Description

Affine transformations are of type $f(x) = Ax + b$, where x is the spatial coordinates (2D in this case), A is a rotation matrix (it can also include scale/shear parameters, but only rotation is considered here), and b is the translation (xy shift) parameters.

Usage

```
affineBrick(Brick, angle = 0, xy_shift = c(0, 0))
```

Arguments

Brick	An object of class RasterBrick, RasterStack or RasterLayer (from package raster).
angle	A numeric value of the angle (in degrees, 0-360) to rotate Brick. A negative value will change the direction of rotation to clockwise.
xy_shift	A numeric vector of length two with the x and y shift (the translation parameters).

Value

An object of the same class as the input Brick.

Note

Affine transformation affects the image dimension.

See Also

affineCoords(), registerBrick()

Examples

```
p <- system.file('exdata', 'soybean.tif', package = 'hyperbrick')
im <- brick(p)
print(im)

# view band-3
plot(im[[3]], col = gray.colors(20), asp = 0)

# rotate band-3 at 3.5 degrees counter-clockwise
b3_rot <- affineBrick(im[[3]], angle = 3.5)
plot(b3_rot, add = TRUE, legend = FALSE,
    col = adjustcolor(terrain.colors(20), 0.5))
```

`affineCoords` *Affine Transformation (rotation and shift) on Spatial Coordinates*

Description

Affine transformations are of type $f(x) = Ax + b$, where x is the spatial coordinates (2D in this case), A is a rotation matrix (it can also include scale/shear parameters, but only rotation is considered here), and b is the translation (xy shift) parameters.

Usage

```
affineCoords(s, angle = 0, xy_shift = c(0, 0))
```

Arguments

`s`	A spatial object where `raster::coordinates()` can be extracted from. Example classes: `Extent`, `RasterLayer`, `RasterBrick`, `RasterStack` (from package raster), `SpatialPolygon` (from package sp).
`angle`	A numeric value of the angle (in degrees, 0-360) to rotate `s`. A negative value will change the direction of rotation to clockwise.
`xy_shift`	A numeric vector of length two with the x and y shift (the translation parameters).

Value

A two-column matrix with the transformed coordinates (xy).

See Also

`affineBrick()`

Examples

```
p <- system.file('exdata', 'soybean.tif', package = 'hyperbrick')
im <- brick(p)
print(im)

# view band-3
plot(im[[3]], col = gray.colors(20), asp = 0)

# draw a spatial polygon on image
pol <- Polygon(extent(c(40, 85, 50, 150)))
lines(pol)

# rotate and shift the spatial polygon
new_pol <- affineCoords(pol, angle = -3, xy_shift = c(-11, 0))
plot(im[[3]], col = gray.colors(20), asp = 0)
lines(new_pol)
```

```
# do some analysis within it, like:
new_pols <- SpatialPolygons(list(Polygons(list(Polygon(new_pol)), "id0")))
plot(mask(im[[3]], new_pols))
mean(extract(im[[3]], new_pols)[[1]])
```

---

buildBrick *Read and Pre-Process Hyperspectral Image*

Description

Retrieve the raw data from the ENVI file, simultaneously read the header file and build a Brick containing all the layers (spectral bands) of the hyperspectral image. Optionally, do pre-processing steps on row data, such as: radiometric correction using the Dark Object Subtraction (DOS) method, set up the spatial extents using the coordinates of a reference layer and inputs of the camera, convert the row values (digital numbers) to spectral features such as radiance or reflectance.

Usage

```
buildBrick(
  path,
  path_hdr = sub(".dat", ".hdr", path, fixed = TRUE),
  hFOV = NULL,
  vFOV = NULL,
  height = NULL,
  ref_layer = 1,
  spectral_feature = c("raw", "radiance", "reflectance"),
  reflectance_method = c("irradiance", "white_panel"),
  dark_path = NULL,
  dark_quantile = 0.25,
  white_path = NULL
)
```

Arguments

path	A character giving the path for the ENVI file (.dat).
path_hdr	(Optional, character) The path for the header file (.hdr).
hFOV	(Optional, numeric) The horizontal Field Of View (in degrees) of the camera. Note: this is used to calculate the spatial extent. See Details.
vFOV	(Optional, numeric) The vertical Field Of View (in degrees) of the camera. Note: this is used to calculate the spatial extent. See Details.
height	(Optional, numeric) The flight altitude (in meters). Note: this is be used to calculate the spatial extent. See Details.
ref_layer	(Optional, integer) The reference layer (spectral band) to be used to set up the geographic location of the image.

`spectral_feature`
: A character giving the name of the spectral feature to be loaded. Must be one of the three: `"raw"` (digital numbers), `"radiance"`, `"reflectance"`. See Details.

`reflectance_method`
: A character for selecting the method to calculate the reflectance values. It must be one of the following: `"irradiance"` or `"white_panel"`. See Details.

`dark_path`
: (Optional) A character giving the path for the ENVI file containing the dark reference raw values, a.k.a. noisy energy, for DOS calibration. See Details.

`dark_quantile`
: A numeric value used to calculate the quantile of the dark reference raw values of each spectral band. Default is 0.25. This will be used for the DOS radiometric correction.

`white_path`
: (Optional) A character giving the path for the ENVI file containing the white panel. See Details.

Details

The geographical coordinates (xy) registered in the header file are automatically retrieved. If the arguments `hFOV`, `vFOV` and `height` are passed, `buildBrick` will automatically compute the UTM zone and the spatial extent, and set up the coordinate reference system.

`"radiance"` is obtained by multiplying the raw values of each spectral band by the respective "gain" values registered in the header file (if available).

If `"reflectance"` is passed as value for the argument `spectral_feature`, then the value of the next argument, `reflectance_method`, will be used to calculate the reflectance values. The `"irradiance"` method consists of using the solar irradiance values registered in the header file (if available) at each spectral band as reference for the radiance values reaching the camera sensor. If `"white_panel"` is chosen, then the path for the ENVI file containing the image of the white panel must be passed as value for the respective argument. Note that `buildBrick` will consider value of the white panel as the maximum.

Make sure that the header file (.hdr) has the same name as the ENVI (.dat) file when pointing them at the arguments `dark_path` or `white_path`.

Value

A `RasterBrick` object (from the package raster).

See Also

`read_hdr_envi()`, `raster::brick()`

Examples

```
# Point to an ENVI data
path <- system.file('exdata', 'obory.dat', package = 'hyperbrick')
# First, let's check the header file
# There are 81 bands. First two are noisy.
# Irradiance is available.
path_hdr <- system.file('exdata', 'obory.hdr', package = 'hyperbrick')
readLines(path_hdr)
```

```
# Example 1 - raw values
path <- system.file('exdata', 'obory.dat', package = 'hyperbrick')
b <- buildBrick(path)
print(b)
plot(b, 35)

# Example 2 - set up CRS and compute radiance
br <- buildBrick(path, hFOV = 36.8, vFOV = 36.8, height = 45,
                ref_layer = 35, spectral_feature = 'radiance')
print(br)
plot(br, 35)

# Example 3 - DOS correction
dpath <- system.file('exdata', 'obory_dark.dat', package = 'hyperbrick')
brd <- buildBrick(path, hFOV = 36.8, vFOV = 36.8, height = 45,
                  ref_layer = 35, spectral_feature = 'radiance',
                  dark_path = dpath)
print(brd)
plot(brd, 35)

# Example 4 - compute reflectance
bre <- buildBrick(path, hFOV = 36.8, vFOV = 36.8, height = 45,
                  ref_layer = 35, spectral_feature = 'reflectance',
                  reflectance_method = "irradiance",
                  dark_path = dpath)
print(bre)
plot(bre, 35)

# Example 5 - there is a white-reference panel in this image
idmax <- which.max(bre[[35]])
plot(bre, 35, asp = 0)
points(xyFromCell(bre[[35]], idmax), pch = 3)

bre2 <- buildBrick(path, hFOV = 36.8, vFOV = 36.8, height = 45,
                   ref_layer = 35, spectral_feature = 'reflectance',
                   reflectance_method = "white_panel",
                   white_path = path, dark_path = dpath)
print(bre2)
plot(bre2, 35)
```

---

read_hdr_envi *Read Metadata from an ENVI Header File*

Description

The ENVI image files are accompanied by an ASCII header file (.hdr format) containing the metadata of the image, such as number of samples (rows), lines (columns), number of spectral bands, wavelength, byte order, data type, gain values (if available), irradiance (if available), coordinates etc. This file is necessary to properly read the image data.

Usage

```
read_hdr_envi(path, hFOV = NULL, vFOV = NULL, height = NULL)
```

Arguments

path	A character giving the path for the header file (.hdr).
hFOV	(Optional, numeric) The horizontal Field Of View (in degrees) of the sensor. Note: this is used to calculate the spatial extent. See Details.
vFOV	(Optional, numeric) The vertical Field Of View (in degrees) of the sensor. Note: this is used to calculate the spatial extent. See Details.
height	(Optional, numeric) The flight altitude (in meters). Note: this is be used to calculate the spatial extent. See Details.

Details

The geographical coordinates (xy) registered in the header file are automatically retrieved. Please be aware that `read_hdr_envi` uses the string "gps" to search this field in the header file. If necessary, rename this field in the header file.

If the arguments `hFOV`, `vFOV` and `height` are passed, `read_hdr_envi` will automatically compute the UTM zone and the spatial extent, and set up the coordinate reference system.

Value

A `list` of the following:

dim An integer vector with the dimensions of the image: number of columns (x), rows (y) and layers (spectral bands).

wavelength A numeric vector of the wavelength registered, the same length as the number of layers (spectral bands).

gain A numeric vector of gain values of each spectral band.

irradiance A numeric vector of the solar irradiance values registered by the corresponding sensor at each spectral band. Please check the unit (usually W/(m^2 μm sr)) with the manufacturer specifications.

coordinates A numeric matrix with the geographic coordinates (xy) registered by the sensor at each spectral band.

extents A four-column numeric matrix with the spatial extents (xmin, xmax, ymin, ymax), in UTM, of each spectral band. This is `NULL` if the optional arguments are not passed.

CRS The Coordinate Reference System of the spatial extents. This is `NULL` if the optional arguments are not passed.

See Also

`buildBrick()`

Examples

```
# Example 1
path_hdr <- system.file('exdata', 'obory.hdr', package = 'hyperbrick')
readLines(path_hdr)
read_hdr_envi(path_hdr)

# Example 2 - set up the CRS to UTM and retrieve extents
read_hdr_envi(path_hdr, hFOV = 36.8, vFOV = 36.8, height = 45)
```

`registerBand` *Single Band-to-Band Registration (Translation)*

Description

Hyperspectral image acquisition normaly causes spatial misalignment between the spectral bands (layers) due to both equipment (such as band-to-band recording delay) and external factors (e.g. sensor vibrations). In this case, a geometric correction is necessary for remote sensing applications such as combining/merging spectral bands. This function uses the HOG (Histogram of Oriented Gradient) descriptor in order to find the optimal translation (xy shift) on a 'slave' band to be spatially align with a 'master' (reference) band.

Usage

```
registerBand(slave, master, ncells = 24, orient = 8)
```

Arguments

`slave`	An object of class `RasterLayer` (from package raster).
`master`	An object of class `RasterLayer` (from package raster).
`ncells`	An integer giving the number of cells to compute the oriented gradients of the HOG descriptor. Default is 24. See `OpenImageR::HOG()`.
`orient`	An integer giving the number of orientations to compute the oriented gradients of the HOG descriptor. Default is 8. See `OpenImageR::HOG()`.

Details

The affine parameters are estimated using a general optimization algorithm. This function only estimates translation parameters. To register bands also with rotation fixes, please check `registerBand3()`. But this should be used carefully, as rotation affects the spatial dimensions.

Value

An object of the same classe as the input `slave`, with the fixed extent. An additional attribute called `'affine_pars'` is stored, containing the shift in x and y, in the same unit as the spatial extent of the image.

See Also

OpenImageR::HOG(), registerBrick(), registerBand3()

Examples

```
# load an image
path <- system.file('exdata', 'obory.dat', package = 'hyperbrick')
dpath <- system.file('exdata', 'obory_dark.dat', package = 'hyperbrick')
im <- buildBrick(path, hFOV = 36.8, vFOV = 36.8, height = 45,
                ref_layer = 35, spectral_feature = 'radiance',
                dark_path = dpath)
print(im)

# check bands 11 (550 nm) and 35 (670 nm)
plot(im[[35]], col = gray.colors(20), asp = 0)
plot(im[[11]], add = TRUE, legend = FALSE,
    col = adjustcolor(heat.colors(20), 0.3))

# register band 11 to band 35
new11 <- registerBand(slave = im[[11]], master = im[[35]])
plot(im[[35]], col = gray.colors(20), asp = 0)
plot(new11, add = TRUE, legend = FALSE,
    col = adjustcolor(heat.colors(20), 0.3))

# see the xy shift on band 11
attr(new11, "affine_pars")
```

`registerBand3` *Single Band-to-Band Registration (Rotation and Translation)*

Description

Hyperspectral image acquisition normaly causes spatial misalignment between the spectral bands (layers) due to both equipment (such as band-to-band recording delay) and external factors (e.g. sensor vibrations). In this case, a geometric correction is necessary for remote sensing applications such as combining/merging spectral bands. This function uses the HOG (Histogram of Oriented Gradient) descriptor in order to find the optimal rotation angle and translation (xy shift) on a 'slave' band to be spatially align with a 'master' (reference) band.

Usage

```
registerBand3(slave, master, ncells = 24, orient = 8, start_affine)
```

Arguments

`slave`	An object of class `RasterLayer` (from package raster).
`master`	An object of class `RasterLayer` (from package raster).

ncells	An integer giving the number of cells to compute the oriented gradients of the HOG descriptor. Default is 24. See OpenImageR::HOG().
orient	An integer giving the number of orientations to compute the oriented gradients of the HOG descriptor. Default is 8. See OpenImageR::HOG().
start_affine	A numeric vector containing the starting values for the affine parameters to be optimized, i.e., the shift in x and y and the rotation angle (in degrees). Example: start_affine = c(1, 0, -2), which indicates a shift of 1 in the x-axis, 0 in the y-axis and a clockwise (negative values) angle of 2 degrees. See more in affineBrick().

Details

This should be used carefully, as rotation affects the spatial dimensions. It is recommended to try registerBand() first.

The affine parameters are estimated using a general optimization algorithm.

Value

An object of the same classe as the input slave, with the fixed extent. An additional attribute called 'affine_pars' is stored, containing the rotation angle (degrees) and the shift in x and y in the same unit as the spatial extent of the image.

See Also

OpenImageR::HOG(), registerBrick(), registerBand()

Examples

```
p <- system.file('exdata', 'soybean.tif', package = 'hyperbrick')
im <- brick(p)
print(im)

# see how layer 1 is misregistered
plot(im[[3]], col = gray.colors(20), asp = 0)
plot(im[[1]], add = TRUE, legend = FALSE,
     col = adjustcolor(terrain.colors(20), 0.6))

# remove the #s to run
# b1_reg <- registerBand3(slave = im[[1]], master = im[[3]],
#                         start_affine = c(0, 0, -2.5))
# attr(b1_reg, "affine_pars")

# plot(im[[3]], col = gray.colors(20), asp = 0)
# plot(b1_reg, add = TRUE, legend = FALSE,
#      col = adjustcolor(terrain.colors(20), 0.6))
```

registerBrick *Hyperspectral Image Registration (Translation)*

Description

Hyperspectral image acquisition normaly causes spatial misalignment between the spectral bands (layers) due to both equipment (such as band-to-band recording delay) and external factors (e.g. sensor vibrations). In this case, a geometric correction is necessary for remote sensing applications such as combining/merging spectral bands. This function uses the HOG (Histogram of Oriented Gradient) descriptor in order to find the optimal translations (xy shift) on multiple 'slave' bands to be spatially align with a 'master' (reference) band. Parallel processing is allowed.

Usage

```
registerBrick(
  Brick,
  ref_layer = 1,
  layers = "all",
  ncells = 24,
  orient = 8,
  cl = NULL
)
```

Arguments

Brick	An object of class RasterBrick or RasterStack (from package raster), containing multiple layers (spectral bands).
ref_layer	An integer indicating which layer (spectral band) should be used as reference ('master') to register all the others from. Default is 1, the first band.
layers	Either the character "all" (default), which indicates that all bands should be registered using ref_layer as reference, or a vector of integers specifying which bands to register.
ncells	An integer giving the number of cells to compute the oriented gradients of the HOG descriptor. Default is 24. See OpenImageR::HOG().
orient	An integer giving the number of orientations to compute the oriented gradients of the HOG descriptor. Default is 8. See OpenImageR::HOG().
cl	An integer indicating the number of parallel processes or an object created by parallel::makeCluster(). Default if NULL.

Details

This should be used carefully, as rotation affects the spatial dimensions. The affine parameters are estimated using a general optimization algorithm.

Value

An object of the same classe as the input slave, with the fixed extent.

See Also

OpenImageR::HOG(), registerBand(), registerBand3(), buildBrick()

Examples

```
path <- system.file('exdata', 'obory.dat', package = 'hyperbrick')
dpath <- system.file('exdata', 'obory_dark.dat', package = 'hyperbrick')
im <- buildBrick(path, hFOV = 36.8, vFOV = 36.8, height = 45,
                 ref_layer = 35, spectral_feature = 'radiance',
                 dark_path = dpath)
print(im)
plotRGB(im, r = 63, g = 34, b = 11, stretch = 'lin')

imreg <- registerBrick(im, ref_layer = 35, layers = c(63, 34, 11))
imreg
plotRGB(imreg, stretch = 'lin')
```

---

slideBrick *Sliding-Windows Statistics Over a Hyperspectral Image*

Description

Calculate focal statistics on a hyperspectral image using non-overlapping sliding windows.

Usage

```
slideBrick(Brick, slide_windows, fun = median)
```

Arguments

Brick	An object of class RasterBrick or RasterStack (from package raster), containing multiple layers (spectral bands).
slide_windows	An object of class slideWindows, which consists of a list of spatial extents giving the location of the non-overlapping sliding windows that covers the entire image.
fun	A vectorized function indicating the statistics to be calculated within each sliding-windows, e.g. median (default), mean, sd.

Value

A vector or matrix containing the estimates of each sliding windows.

See Also

slideWindows()

Examples

```
p <- system.file('exdata', 'obory.dat', package = 'hyperbrick')
im <- buildBrick(p, ref_layer = 35,
                 spectral_feature = "radiance",
                 hFOV = 36.8, vFOV = 36.8, height = 45)
print(im)
plotRGB(im, r = 63, b = 34, g = 11, scale = 90)

ext <- extent(c(512700.2, 512715, 5769462, 5769477))
sw <- slideWindows(ext, n = c(7, 7))
lapply(sw, lines, col = "white") -> null_obj

sb <- slideBrick(im, sw, fun = mean)
head(sb)
```

`slideWindows` *Create Sliding Windows Over a Spatial Object*

Description

Create a prefixed number of non-overlapping sliding windows that cover the entire extent of a spatial object.

Usage

```
slideWindows(x, n = c(8, 8))
```

Arguments

`x`	A spatial object where an extent can be extracted from. Classes: `Extent`, `RasterLayer`, `RasterBrick`, `RasterStack` (from package raster).
`n`	An integer vector of length two determining the number of divisions in x and y directions. Default is `n = c(8, 8)`, which creates 64 windows.

Value

An object of class `slideWindows`, which consists of a list of each sliding-windows extent.

See Also

`slideBrick()`, raster::extent

Examples

```
# Example 1
p <- system.file('exdata', 'soybean.tif', package = 'hyperbrick')
im <- brick(p)
plotRGB(im)

sw <- slideWindows(im, n = c(8, 8))
lapply(sw, lines) -> null_obj
lines(sw[[1]], col = "white", lwd = 3)
lines(sw[[64]], col = "white", lwd = 3)

# Example 2
ext <- extent(c(30, 350, 150, 230))
lines(ext, col = "red", lwd = 3)
sw2 <- slideWindows(ext, n = c(18, 6))
lapply(sw2, lines, col = "red") -> null_obj
```

`viewSpectra` *Spectral Signature of a Hyperspectral Image*

Description

Visualize statistics calculated through the bands of a hyperspectral image.

Usage

```
viewSpectra(x, ...)
```

Arguments

`x`	A numeric matrix or vector containing the values to be plotted at each spectral band (wavelength). Generally, an object obtained with `slideBrick()`.
`...`	Further graphical parameters. See `par()`.

See Also

`slideBrick()`

Examples

```
p <- system.file('exdata', 'obory.dat', package = 'hyperbrick')
im <- buildBrick(p, ref_layer = 35,
                 spectral_feature = "radiance",
                 hFOV = 36.8, vFOV = 36.8, height = 45)
plotRGB(im, r = 63, b = 34, g = 11, scale = 90, axes = TRUE)

sw <- slideWindows(im)
lapply(sw, lines, col = "white") -> null_obj
```

```
sb <- slideBrick(im, sw, fun = mean)
head(sb)

viewSpectra(sb, ylab = "Radiance")
```

Index