Package 'uavRst'

November 19, 2018

Type Package

Title Unmanned Aerial Vehicle Remote Sensing Tools

Version 0.5-0

Date 2018-11-11

Encoding UTF-8

Maintainer Chris Reudenbach <reudenbach@uni-marburg.de>

Description

Support the analysis of drone derived imagery and point clouds as a cheap and easy to use alternative/complement to light detection and ranging data. It provides functionality to analyze poor quality digital aerial images as taken by low budget ready to fly drones. This includes supported machine learning based classification functions, comprehensive texture analysis, segmentation algorithms as well as forest relevant analyzes of metrics and measures on the derived products.

License GPL (>= 3) | file LICENSE

Depends R (>= 3.1.0)

Imports raster, foreach

Suggests knitr, stringr, sp, sf, htmlwidgets, htmltools, Rcpp, rgdal, rgeos, gdalUtils, tools, caret, zoo, data.table, parallel, spatial.tools, velox, link2GI, doParallel, CAST, glcm, crayon, ForestTools, itcSegment, pROC, methods, RSAGA, reshape2, rgrass7, randomForest, rLiDAR, rlas, lidR, rmarkdown, mapview

LinkingTo Rcpp

RoxygenNote 6.1.1

SystemRequirements GNU make

NeedsCompilation yes

Author Florian Detsch [ctb],
Hanna Meyer [aut],
Thomas Nauss [ctb],
Lars Opgenoorth [ctb],
Chris Reudenbach [cre, aut],
Environmental Informatics Marburg [ctb]

Repository CRAN

Date/Publication 2018-11-19 19:00:03 UTC

R topics documented:

calc_ext *Convenient function to preprocess synthetic raster bands from a given RGB image and/or DTM/DSM data.*

Description

Convenient function to preprocess synthetic raster bands from a given RGB image and/or DTM/DSM data. Optionally the raster values can be extracted to data frames on base of vector data. This is useful for for classification training purposes and covers usually step 1 of the random forest based classification of UAV derived visible imagery and point clouds.

Usage

```
calc_ext(calculateBands = FALSE, extractTrain = TRUE,
  prefixRun = "temp", patterndemFiles = "dem", prefixTrainImg = "",
  prefixTrainGeom = "", suffixTrainImg = "", suffixTrainGeom = "",
  patternIdx = "index", patternImgFiles = "", channels = c("red",
  "green", "blue"), hara = TRUE, haraType = c("simple", "advanced",
  "higher"), stat = TRUE, edge = TRUE, edgeType = c("gradient",
  "sobel", "touzi"), morpho = TRUE, morphoType = c("dilate", "erode",
  "opening", "closing"), rgbi = TRUE, indices = c("VVI", "VARI",
  "NDTI", "RI", "SCI", "BI", "SI", "HI", "TGI", "GLI", "NGRDI", "GRVI",
  "GLAI", "HUE", "CI", "SAT", "SHP"), rgbTrans = TRUE,
  colorSpaces = c("CIELab", "CMY", "Gray", "HCL", "HSB", "HSI", "Log",
  "XYZ", "YUV"), pardem = TRUE, demType = c("hillshade", "slope",
  "aspect", "TRI", "TPI", "Roughness", "SLOPE", "ASPECT", "C_GENE",
  "C_PROF", "C_PLAN", "C_TANG", "C_LONG", "C_CROS", "C_MINI", "C_MAXI",
  "C_TOTA", "C_ROTO", "MTPI"), morphoMethod = 6, minScale = 1,
  maxScale = 8, numScale = 2, kernel = 3, currentDataFolder = NULL,
  currentIdxFolder = NULL, cleanRunDir = TRUE, giLinks = NULL)
```

Arguments

calculateBands	logical. switch for set on calculation of syntheic bands and indices default = TRUE
extractTrain	logical. switch for set on extract training data according to training geometries default = TRUE
prefixRun	character. general prefix of all result files of the current run default = "tmp"
patterndemFiles	character. mandantory if DEM data is processes. prefix of current DEM default = "dem"
prefixTrainImg	character. potential string of characters that is used in the beginning of a shape file prefixTrainImg_SHAPEFILENAME_suffixTrainImg
prefixTrainGeom	character. potential string of characters that is used in the beginning of a shape file prefixTrainGeom_SHAPEFILENAME_suffixTrainGeom

suffixTrainImg	character. potential string of characters that is used in the beginning of a shape file prefixTrainImg_SHAPEFILENAME_suffixTrainImg
suffixTrainGeom	character. potential string of characters that is used in the beginning of a shape file prefixTrainGeom_SHAPEFILENAME_suffixTrainGeom
patternIdx	character. code string that will concatenated to the filename to identify the index file
patternImgFiles	character. mandantory string that exist in ech imagefile to be processes
channels	character. channels to be choosed options are c("red", "green", "blue") default = c("red", "green", "blue")
hara	logical. switch for using HaralickTextureExtraction, default = TRUE.
haraType	character. hara options, default is c("simple"), other options are "advanced" "higher" "all". NOTE: "higher" takes a LOT of time
stat	logical. switch for using statistic default = TRUE the stas are mean,variance, curtosis, skewness
edge	logical. switch for using edge filtering default = TRUE
edgeType	character. edge options, default is c("gradient","sobel","touzi") all options are c("gradient","sobel","touzi")
morpho	logical. switch for using morphological filtering default = TRUE
morphoType	character. morphological options, default is c("dilate","erode","opening","closing") all options are ("dilate","erode","opening","closing")
rgbi	logical. switch for using rgbi index calcualtions default = TRUE
indices	character. RGB indices, default is c("VARI","NDTI","RI","CI","BI","SI","HI","TGI","GLI","NGRDI") all options are c("VVI","VARI","NDTI","RI","SCI","BI","SI","HI","TGI","GLI","NGRDI","GRVI","GL
rgbTrans	logical. switch for using color space transforming default = TRUE
colorSpaces	character. RGB colorspace transforming to default c("CIELab","CMY","Gray","HCL","HSB","HSI","Lo
pardem	logical. switch for calculating dem parameter, default = FALSE
demType	character. ("hillshade","slope", "aspect","TRI","TPI","Roughness")
morphoMethod	numeric. saga morphometric method
minScale	numeric. in scale for multi scale TPI
maxScale	numeric. max scale for multi scale TPI
numScale	numeric. number of scale for multi scale TPI
kernel	numeric. size of kernel for filtering and statistics, default is 3
currentDataFolder	NULL folder to image (and shape) data
currentIdxFolder	NULL folder for saving the results
cleanRunDir	logical. TRUE logical switch for deleting the calculated tifs, default is TRUE
giLinks	list. GI tools cli paths

Details

(01) calc_ext() calculation of spectral indices, basic spatial statistics and textures and extracting of training values over all channels according to training data

(02) ffs_train() training using random forest and the forward feature selection method startTrain=TRUE

(03) calc_ext() with respect to the selected predictor variables you may calculate the requested channels for all rgb data that you want to predict.

(04) prediction startPredict=TRUE

Note

If the function is used for stand alone extraction of the training data please provide both, the imagestack containing the raster data plus the corresponding band names (usually saved as an Rdata file) and the corresponding shape file.

The workflow is intended to use the forward feature selection as decribed by Meyer et al. (2018). This approach needs at least a pair of images that differ in time and/or space for a leave one location out validation mode. You may overcome this situation if you tile your image and provide for each tile seperate training data. If you just want to classify a single image by a single training file use the normal procedure as provided by the `trainControl` function.

Examples

```
## Not run:

##- required packages
require(uavRst)
require(link2GI)

# create and check the links to the GI software
giLinks<-uavRst::linkAll()
if (giLinks$saga$exist & giLinks$otb$exist){

##- set root folder
projRootDir<-tempdir()

##- create and set subfolders
##- please mind that the pathes are exported as global variables
paths<-link2GI::initProj(projRootDir = projRootDir,
                         projFolders = c("data/","data/ref/","output/","run/","las/"),
                         global = TRUE,
                         path_prefix = "path_")

##- overide trailing backslash issue
path_run<-ifelse(Sys.info()["sysname"]=="Windows", sub("/$", "",path_run),path_run)
setwd(path_run)
```

```
##- clean runtime folder
unlink(paste0(path_run,"*"), force = TRUE)

# get the tutorial data
utils::download.file("https://github.com/gisma/gismaData/raw/master/uavRst/data/tutorial_data.zip",
                     paste0(path_run,"tutorial_data.zip"))
unzip(zipfile = paste0(path_run,"tutorial_data.zip"), exdir = path_run)


##- calculate some synthetic channels from the RGB image and the canopy height model
##- then extract the from the corresponding training geometries the data values aka trainingdata
trainDF <- calc_ext(calculateBands    = TRUE,
                    extractTrain      = TRUE,
                    suffixTrainGeom   = "",
                    patternIdx        = "index",
                    patternImgFiles   = "rgb" ,
                    patterndemFiles   = "chm",
                    prefixRun         = "tutorial",
                    prefixTrainImg    = "",
                    rgbi              = TRUE,
                    indices           = c("TGI","CI"),
                    channels          = c("red"),
                    rgbTrans          = FALSE,
                    hara              = FALSE,
                    haraType          = c("simple"),
                    stat              = FALSE,
                    edge              = FALSE,
                    morpho            = FALSE,
                    pardem            = TRUE,
                    demType           = c("slope", "MTPI"),
                    kernel            = 3,
                    currentDataFolder = path_run,
                    currentIdxFolder  = path_run,
                    giLinks = giLinks)

##- show the result
head(trainDF)
# use ffs_train as next step for rf classification issues
}
##+
## End(Not run)
```

---

chmseg_FT *Watershed segmentation based on 'ForestTools'*

Description

'ForestTools' segmentation of individual tree crowns based on a canopy height model and initial seeding points (trees). Very fast algorithm based on the imagr watershed algorithm. Andrew Plowright: R package 'ForestTools'

Usage

```
chmseg_FT(treepos = NULL, chm = NULL, minTreeAlt = 2,
  format = "polygons", winRadius = 1.5, verbose = FALSE)
```

Arguments

treepos	raster*. The positions of the estimated top of trees. The function will generally produce a number of crown segments equal to the number of treetops.
chm	raster*. Canopy height model in raster format. Should be the same that was used to create the input for treepos.
minTreeAlt	numeric. The minimum height value for a CHM pixel to be considered as part of a crown segment. All chm pixels beneath this value will be masked out. Note that this value should be lower than the minimum height of treepos.
format	character. Format of the function's output. Can be set to either 'raster' or 'polygons'.
winRadius	numeric the fixed radius written in the SPDF of the tree top file if not generated by Foresttools
verbose	to be quiet FALSE

Examples

```
 require(uavRst)
## get the data
 data(chm_seg)
 data(trp_seg)
## you may create the Foresttools tree tops
## otherwise the raster is transformed to a SPDF according to the FT format
#trp <- ForestTools::vwf(chm_seg[[1]], winFun = function(x){x * 0.06 + 0.5}, minHeight = 2)
## segmentation
crownsFT <- chmseg_FT(chm = chm_seg[[1]],
                      treepos = trp_seg[[1]],
                      format = "polygons",
                      minTreeAlt = 23,
                      winRadius = 1.5,
                      verbose = FALSE)

## Visualisation
raster::plot(crownsFT)
```

---

chmseg_GWS *seeded region growing tree crown segmentation based on 'SAGA GIS'*

Description

Tree segmentation based on a CHM, basically returns a vector data set with the tree crown geometries and a bunch of corresponding indices. After the segementation itself, the results are hole filled and optionally, it can be filtered by a majority filter.

Usage

```
chmseg_GWS(treepos = NULL, chm = NULL, minTreeAlt = 2,
  minTreeAltParam = "chmQ20", maxCrownArea = 100, leafsize = 256,
  normalize = 0, neighbour = 1, method = 0, thVarFeature = 1,
  thVarSpatial = 1, thSimilarity = 0.002,
  segmentationBands = c("chm"), majorityRadius = 3, parallel = 1,
  giLinks = NULL)
```

Arguments

`treepos`	raster* object
`chm`	raster*. Canopy height model in `raster` format. Should be the same that was used to create the input for `treepos`.
`minTreeAlt`	numeric. The minimum height value for a `chm` pixel is to be considered as part of a crown segment. All `chm` pixels beneath this value will be masked out. Note that this value should be lower than the minimum height of `treepos`.
`minTreeAltParam`	character. code for the percentile that is used as tree height treshold. It is build using the key letters `chmQ` and adding the percentile i.e. "10". Default is `chmQ20`
`maxCrownArea`	numeric. A single value of the maximum individual tree crown radius expected. Default 10.0 m.
`leafsize`	integer. bin size of grey value sampling range from 1 to 256 see also: SAGA GIS Help
`normalize`	integer. logical switch if data will be normalized (1) see also: SAGA GIS Help
`neighbour`	integer. von Neumanns' neighborhood (0) or Moore's (1) see also: SAGA GIS Help
`method`	integer. growing algorithm for feature space and position (0) or feature space only (1), see also: SAGA GIS Help
`thVarFeature`	numeric. Variance in Position Space see also: SAGA GIS Help
`thVarSpatial`	numeric. Variance in Feature Space see also: SAGA GIS Help
`thSimilarity`	mumeric. Similarity Threshold see also: SAGA GIS Help
`segmentationBands`	character. a list of raster data that is used for the segmentation. The canopy height model `c("chm")` is mandantory. see also: SAGA GIS Help
`majorityRadius`	numeric. kernel size for the majority filter out spurious pixel
`parallel`	running parallel or not default = 1
`giLinks`	list. of GI tools cli paths

Author(s)

Chris Reudenbach

Examples

```
## Not run:
##- required packages
require(uavRst)
require(link2GI)
##- linkages
##- create and check the links to the GI software
giLinks<-uavRst::linkAll()
if (giLinks$saga$exist & giLinks$otb$exist & giLinks$grass$exist) {

##- project folder
projRootDir<-tempdir()

##- create subfolders please mind that the pathes are exported as global variables
paths<-link2GI::initProj(projRootDir = projRootDir,
                         projFolders = c("data/","data/ref/","output/","run/","las/"),
                         global = TRUE,
                         path_prefix = "path_")
##- overide trailing backslash issue
 path_run<-ifelse(Sys.info()["sysname"]=="Windows", sub("/$", "",path_run),path_run)
 setwd(path_run)
 unlink(paste0(path_run,"*"), force = TRUE)

##- get the data
 data(chm_seg)
 data(trp_seg)

##- tree segmentation
crowns_GWS <- chmseg_GWS( treepos = trp_seg[[1]],
                      chm = chm_seg[[1]],
                      minTreeAlt = 2,
                      neighbour = 0,
                      thVarFeature = 1.,
                      thVarSpatial = 1.,
                      thSimilarity = 0.00001,
                      giLinks = giLinks )

##- visualize it
mapview::mapview(crowns_GWS,zcol="chmMEAN")
}

## End(Not run)
```

---

chmseg_ITC *Decision tree segmentation method to grow individual tree crowns based on 'itcSegment'*

Description

Segmentation of individual tree crowns as polygons based on a LiDAR derived canopy height model. Michele Dalponte: R package itcSegment. M. Dalponte, F. Reyes, K. Kandare, and D.

Gianelle, "Delineation of Individual Tree Crowns from ALS and Hyperspectral data: a comparison among four methods," European Journal of Remote Sensing, Vol. 48, pp. 365-382, 2015.

Usage

```
chmseg_ITC(chm = NULL, EPSG = 3064, movingWin = 7,
  TRESHSeed = 0.45, TRESHCrown = 0.55, minTreeAlt = 2,
  maxCrownArea = 100)
```

Arguments

chm	raster*, Canopy height model in raster or SpatialGridDataFrame file format. Should be the same that was used to create the input for treepos.
EPSG	character. The EPSG code of the reference system of the CHM raster image.
movingWin	numeric. Size (in pixels) of the moving window to detect local maxima. itcSegment
TRESHSeed	numeric. seeding threshold. itcSegment
TRESHCrown	numeric. crowns threshold. itcSegment
minTreeAlt	numeric. Height threshold (m) below a pixel cannot be a local maximum. Local maxima values are used to define tree tops.itcSegment
maxCrownArea	numeric. A single value of the maximum individual tree crown radius expected. Default 10.0 m. height of treepos.

Examples

```
 require(uavRst)
 data(chm_seg)
##- segmentation
crownsITC<- chmseg_ITC(chm = chm_seg[[1]],
                       EPSG =3064,
                       movingWin = 7,
                       TRESHSeed = 0.45,
                       TRESHCrown = 0.55,
                       minTreeAlt = 5,
                       maxCrownArea = 50)

##- visualisation
raster::plot(crownsITC)
```

---

chmseg_RL *Watershed segmentation based on 'rLiDAR'*

Description

'rLiDAR' segmentation of individual tree crowns based on a canopy height model and initial seeding points (trees). Generic segmentation algorithm Carlos A. Silva et all.: R package rLiDAR

Usage

```
chmseg_RL(treepos = NULL, chm = NULL, maxCrownArea = 150,
  exclusion = 0.2)
```

Arguments

treepos	numeric. matrix or data.frame with three columns (tree xy coordinates and height). number of crown segments equal to the number of treetops.
chm	raster*. Canopy height model in raster or SpatialGridDataFrame file format. Should be the same that was used to create the input for treepos.
maxCrownArea	numeric. A single value of the maximum individual tree crown radius expected. Default 10.0 m. height of treepos.
exclusion	numeric. A single value from 0 to 1 that represents the percent of pixel exclusion.

Examples

```
## Not run:
## required packages
 require(uavRst)
## read chm and tree position data
 data(chm_seg)
 data(trp_seg)
## segmentation
 crownsRL <- chmseg_RL(chm= chm_seg[[1]],
                       treepos= trp_seg[[1]],
                       maxCrownArea = 150,
                       exclusion = 0.2)
## visualisation
 raster::plot(crownsRL)

## End(Not run)
```

---

chm_seg *Canopy height model raster map*

Description

Example data set containing the canopy height map of a small plot sampled in the Maburg University Forest (MOF). The resolution is 10 cm. ETRS89 UTM32

Format

"raster::raster"

colorspace *imagemagick based function to transform the RGB color space into another*

Description

Imagemagick is used for the calculation of the transformations. Please provide a tif file

Usage

```
colorspace(input = NULL, colorspace = c("CIELab", "CMY", "Gray", "HCL",
  "HSB", "HSI", "Log", "XYZ", "YUV"), compress = "None", depth = 8,
  verbose = FALSE, retRaster = TRUE)
```

Arguments

input	character. name of a (Geo)Tiff containing RGB data channels
colorspace	character. For a list of argument to determine colorspace see colorspace manual
compress	character. compression type depending on the imagemagick version the choices are: None, BZip, Fax, Group4, JPEG, JPEG2000, Lossless, LZW, RLE or Zip.
depth	numeric. color space depth in bit default is 8
verbose	be quiet
retRaster	logical if true a rasterstack is returned

Note

you need to install imagemagick on your system, please look at see imagemagick download section.

Examples

```
## Not run:

##- required packages
require(uavRst)
setwd(tempdir())
##- set locale
tmp<-Sys.setlocale('LC_ALL','C')
##- get some typical data as provided by the authority
utils::download.file(url="http://www.ldbv.bayern.de/file/zip/5619/DOP%2040_CIR.zip",
                     destfile="testdata.zip")
unzip("testdata.zip",junkpaths = TRUE,overwrite = TRUE)
##- original color space
raster::plotRGB(raster::stack("4490600_5321400.tif"))
##- change colorspace from RGB to HSI
r2 <- colorspace(input="4490600_5321400.tif",colorspace="HSI")

##- visualize it
```

```
raster::plotRGB(r2)

##- reset locale
tmp<-Sys.setlocale(category = "LC_ALL", locale = "de_DE.-8")
##+
## End(Not run)
```

`crown_filter` *basic filtering of crown polygons using altitude, area and other optional thresholds*

Description

applies basic filtering of crown polygons using altitude, area and other optional thresholds. crown_filter basically returns SPDF with the crown polygons and all calculated parameters.

Usage

```
crown_filter(crownFn, minTreeAlt = 10, minCrownArea = 5,
  maxCrownArea = 100, minTreeAltParam = "chmQ50", crownSTDW = NULL,
  opt = NULL, TAopt = NULL,
  proj4string = "+proj=utm +zone=32 +ellps=GRS80 +units=m +no_defs")
```

Arguments

`crownFn`	filname of OGR compliant vector file
`minTreeAlt`	minimum height in meter that will be regarded as tree
`minCrownArea`	minimum area of crowns that is accepted
`maxCrownArea`	maximum area of crowns that is accepted
`minTreeAltParam`	parameter that is used for filtering MinTreeAlt, default is Median "chmQ50"
`crownSTDW`	parameter that optionally filters for the STDV of the crown altitudes, default is NULL
`opt`	threshold value for optional filter, default is NULL
`TAopt`	optional parameter that might be used for filtering, default is NULL
`proj4string`	proj4 string

Author(s)

Chris Reudenbach

Examples

```
## Not run:

 require(uavRst)
 require(link2GI)

# project folder
 projRootDir<-tempdir()

# create subfolders please mind that the pathes are exported as global variables
 paths<-link2GI::initProj(projRootDir = projRootDir,
                          projFolders = c("data/","data/ref/","output/","run/","las/"),
                          global = TRUE,
                          path_prefix = "path_")
# overide trailing backslash issue
 path_run<-ifelse(Sys.info()["sysname"]=="Windows", sub("/$", "",path_run),path_run)
 setwd(path_run)

# get the data
utils::download.file("https://github.com/gisma/gismaData/raw/master/uavRst/data/crowns.zip",
                       paste0(path_run,"crowns.zip"))
unzip(zipfile = paste0(path_run,"crowns.zip"), exdir = path_run)

# start postclassification of segements
tree_crowns <- crown_filter(crownFn =  paste0(path_run,"polyStat.shp"),
                            minTreeAlt = 3,
                            minCrownArea = 1,
                            maxCrownArea = 150,
                            minTreeAltParam = "chmQ20" )
# visualize it
mapview::mapview(tree_crowns[[2]])
##+
## End(Not run)
```

`digitize` *Easy digitizing of vector features within your rstudio session (or any browser)*

Description

digitize is based on the leaflet draw plugin. It provides a bunch of leaflet maps as base layers for digitizing vector features.

Usage

```
digitize(mapCenter = NULL, zoom = 15, line = TRUE,
  rectangle = TRUE, poly = TRUE, circle = TRUE, point = TRUE,
  remove = TRUE, position = "topright",
```

```
maplayer = c("CartoDB.Positron", "OpenStreetMap", "Esri.WorldImagery",
"Thunderforest.Landscape", "OpenTopoMap"), overlay = NULL,
features = NULL, preset = "all", locPreset = "muf", cex = 10,
lwd = 2, alpha = 0.6, opacity = 0.7)
```

Arguments

mapCenter	c(lat,lon) central point of the leaflet map
zoom	initial zoom level
line	enable the draw tool line tool
rectangle	enable the draw polygon tool
poly	enable the draw polygon tool
circle	enable the draw circle tool
point	enable the draw point tool
remove	enable/disable the remove feature of the draw tool
position	place to put the toolbar (topright, topleft, bottomright, bottomleft)
maplayer	string. as provided by leaflet-provider
overlay	optional sp object
features	features
preset	character. defaut is "NULL" full draw version, "uav" for flightarea digitizing, "ext" for rectangles
locPreset	character. default is "muf" for Marburg University Forest, others are "tra" Trad-delstein, "hag" Hagenstein, "baw" Bayerwald.
cex	cex
lwd	lwd
alpha	alpha
opacity	opacity

Note

You can either save the digitized object to a JSON or KML file to your hard disk. As an alternative you may grab the JSON string via the clipboard.

Author(s)

Chris Reudenbach

Examples

```
## Not run:
##- libs
require(sp)
require(uavRst)

##- preset for digitizing uav flight areas in Meuse
```

```
require(sp)
data(meuse)
sp::coordinates(meuse) <- ~x+y
sp::proj4string(meuse) <-sp::CRS("+init=epsg:28992")
me<-sp::spTransform(meuse,CRSobj = sp::CRS("+init=epsg:4326"))
uavRst::digitize(overlay = me)

##- preset for digitizing extents
uavRst::digitize(preset="ext",overlay = me)
## End(Not run)
```

`ffs_train` *Forward feature selection based on rf model*

Description

ffs_train is a wrapper function for a simple use of the forward feature selection approach of training random forest classification models. This validation is particulary suitable for leave-location-out cross validations where variable selection MUST be based on the performance of the model on the hold out station. See Meyer et al. (2018) for further details. This is in fact the case while using time space variable vegetation patterns for classification purposes. For the UAV based RGB/NIR imagery, it provides an optimized preconfiguration for the classification goals.

Usage

```
ffs_train(trainingDF = NULL, predictors = c("R", "G", "B"),
  response = "ID", spaceVar = "FN", names = c("ID", "R", "G", "B",
  "A", "FN"), noLoc = NULL, sumFunction = "twoClassSummary",
  pVal = 0.5, prefin = "final_", preffs = "ffs_",
  modelSaveName = "model.RData", runtest = FALSE, seed = 100,
  withinSE = TRUE, mtry = 2, noClu = 1)
```

Arguments

`trainingDF`	dataframe. containing training data
`predictors`	character. vector of predictor names as given by the header of the training data table
`response`	character. name of response variable as given by the header of the training data table
`spaceVar`	character. name of the spacetime splitting vatiable as given by the header of the training data table
`names`	character. all names of the dataframe header
`noLoc`	numeric. number of locations to leave out usually number of discrete trainings locations/images
`sumFunction`	character. function to summarize default is "twoClassSummary"

pVal	numeric. used part of the training data default is 0.5
prefin	character. name pattern used for model default is "final_"
preffs	character. name pattern used for ffs default is "ffs_"
modelSaveName	character. name pattern used for saving the model default is "model.RData"
runtest	logical. default is false, if set a external validation will be performed
seed	numeric. number for seeding
withinSE	locical. compares the performance to models that use less variables (e.g. if a model using 5 variables is better than a model using 4 variables but still in the standard error of the 4-variable model, then the 4-variable model is rated as the better model).
mtry	numerical. Number of variable is randomly collected to be sampled at each split time
noClu	numeric. number of cluster to be used

Note

The workflow of uavRst is intended to use the forward feature selection as decribed by Meyer et al. (2018). This approach needs at least a pair of images that differ in time and/or space for a leave one location out validation mode. You may overcome this situation if you tile your image and provide for each tile seperate training data. If you just want to classify a single image by a single training file use the normal procedure as provided by the trainControl function.

Examples

```
## Not run:
require(uavRst)

##- project folder
projRootDir<-tempdir()

# create subfolders please mind that the pathes are exported as global variables
paths<-link2GI::initProj(projRootDir = projRootDir,
                         projFolders = c("data/","data/ref/","output/","run/","las/"),
                         global = TRUE,
                         path_prefix = "path_")
##- overide trailing backslash issue
path_run<-ifelse(Sys.info()["sysname"]=="Windows", sub("/$", "",path_run),path_run)
setwd(path_run)
unlink(paste0(path_run,"*"), force = TRUE)

##- get the rgb image, chm and training data
utils::download.file("https://github.com/gisma/gismaData/raw/master/uavRst/data/ffs.zip",
                     paste0(path_run,"ffs.zip"))
unzip(zipfile = paste0(path_run,"ffs.zip"),exdir = path_run)

##- get geometrical training data assuming that you have used before the calc_ext function
trainDF<-readRDS(paste0(path_run,"tutorial_trainDF.rds"))
load(paste0(path_run,"tutorialbandNames.RData"))
```

```
##- define the classes
 idNumber=c(1,2,3)
 idNames= c("green tree","yellow tree","no tree")
##- add classes names
 for (i in 1:length(idNumber)){
   trainDF$ID[trainDF$ID==i]<-idNames[i]
 }
##- convert to factor (required by rf)
 trainDF$ID <- as.factor(trainDF$ID)
##- now prepare the predictor and response variable names
##- get actual name list from the DF
 name<-names(trainDF)
##- cut leading and tailing ID/FN
 predictNames<-name[3:length(name)-1]

##- call Training
 model <-  ffs_train(trainingDF= trainDF,
                     predictors= predictNames,
                     response  = "ID",
                     spaceVar  = "FN",
                     names     = name,
                     pVal      = 0.1,
                     noClu     = 1)

##- for classification/prediction go ahead with the predict_RGB function
##+
## End(Not run)
```

`get_counts` *counts pixel values according to their classes*

Description

counts pixel values according to their classes

Usage

```
get_counts(ids = c(1, 2), position = NULL, imageFiles = NULL,
  buffersize = 1.5, outPrefix = "classified_index_", ext = ".tif",
  path = path_output, dropChars = 0)
```

Arguments

`ids`	numeric. the ids used for the training
`position`	sp. spatialpoint object containing the centre target positions
`imageFiles`	raster* image/classification file
`buffersize`	numeric. radius in meters around position
`outPrefix`	character. out prefix string

ext	character. extension
path	character. output path
dropChars	numeric. number of characters that should be dropped at the end of the filename

Examples

```
## Not run:
##- required packages
 require(uavRst)
 require(link2GI)

##- project root folder
 projRootDir<-tempdir()

##- create subfolders please mind that the pathes are exported as global variables
 paths<-link2GI::initProj(projRootDir = projRootDir,
                          projFolders = c("data/","data/ref/","output/","run/","las/"),
                          global = TRUE,
                          path_prefix = "path_")
##- overide trailing backslash issue
 path_run<-ifelse(Sys.info()["sysname"]=="Windows", sub("/$", "",path_run),path_run)
 setwd(path_run)
 unlink(paste0(path_run,"*"), force = TRUE)

##- get the rgb image, chm and training data
 utils::download.file("https://github.com/gisma/gismaData/raw/master/uavRst/data/tutorial.zip",
                      paste0(path_run,"tutorial_data.zip"))
 unzip(zipfile = paste0(path_run,"tutorial_data.zip"),exdir = path_run)

# read data
 position <- raster::shapefile(paste0(path_run,"position.shp"))
 imageFiles <-Sys.glob(paths = paste0(path_run,"rgb*","tif"))
 imageTrainStack<-lapply(imageFiles, FUN=raster::stack)

## get counts
 df1<-get_counts(position = position,
                       ids = c(100,200),
               imageFiles = imageFiles,
                outPrefix = "",
                      ext = ".tif",
                     path = path_run)
##+
## End(Not run)
```

---

get_traindata *Extracts training data from a raster stack using vector data as a mask.*

Description

Extracts training data from a raster stack and returns a dataframe containing for each pixel all values.

Usage

```
get_traindata(rasterStack = NULL, trainPlots = NULL)
```

Arguments

rasterStack	an object of rasterstack*. containing image data to make prediction on
trainPlots	an object of SpatialPolygonDataFrame*. providing the training areas

Author(s)

Chris Reudenbach

Examples

```
## Not run:
##- required packages
require(uavRst)
require(link2GI)

##- project folder
projRootDir<-tempdir()

##- create subfolders please mind that the pathes are exported as global variables
paths<-link2GI::initProj(projRootDir = projRootDir,
                         projFolders = c("data/","data/ref/","output/","run/","las/"),
                         global = TRUE,
                         path_prefix = "path_")
##- overide trailing backslash issue
path_run<-ifelse(Sys.info()["sysname"]=="Windows", sub("/$", "",path_run),path_run)
setwd(path_run)
unlink(paste0(path_run,"*"), force = TRUE)

##- get the tutorial data
utils::download.file("https://github.com/gisma/gismaData/raw/master/uavRst/data/tutorial_data.zip",
                     paste0(path_run,"tutorial_data.zip"))
unzip(paste0(path_run,"tutorial_data.zip"),exdir =  path_run)

##- get the files
imageTrainStack <- list()
geomTrainStack <- list()
imageTrainFiles <- Sys.glob(paste0(path_run,"rgb*.tif"))
geomTrainFiles <- Sys.glob(paste0(path_run,"rgb*.shp"))

##- create stacks from image and geometry files
imageTrainStack<-lapply(imageTrainFiles, FUN=raster::stack)
geomTrainStack  <- lapply(geomTrainFiles, FUN=raster::shapefile)
names(imageTrainStack[[1]])<-c("red","green","blue")
names(imageTrainStack[[2]])<-c("red","green","blue")

##' finally extraxt the training data to a data frame
trainDF <- get_traindata(rasterStack  = imageTrainStack,
```

```
                        trainPlots = geomTrainStack)

##- have a look at the training data
head(trainDF)

## End(Not run)
```

---

glcm_texture *Calls the glcm package with useful settings*

Description

Calls the glcm package with useful settings

Usage

```
glcm_texture(x, nrasters = 1:raster::nlayers(x), kernelSize = c(3),
  stats = c("mean", "variance", "homogeneity", "contrast",
  "dissimilarity", "entropy", "second_moment", "correlation"),
  shift = list(c(0, 1), c(1, 1), c(1, 0), c(1, -1)), parallel = TRUE,
  n_grey = 8, min_x = NULL, max_x = NULL)
```

Arguments

x	rasterLayer or a rasterStack containing different channels
nrasters	vector of channels to use from x. Default =nlayers(x)
kernelSize	vector of numbers indicating the environment sizes for which the textures are calculated
stats	string vector of parameters to be calculated.
shift	=list(c(0,1), c(1,1), c(1,0),c(1,-1))
parallel	logical value indicating whether parameters are calculated parallel or not
n_grey	number of grey values.
min_x	for each channel the minimum value which can occur. If NULL then the minimum value from the rasterLayer is used.
max_x	for each channel the maximum value which can occur. If NULL then the maximum value from the rasterLayer is used. This functions calls the glcm function from with standard settings and returns list of RasterStacks containing the texture parameters for each combination of channel and kernelSize

Note

for the use of glcm_texture a glcm wrapper function a raster* object is required

More information at: texture tutorial Keep in mind that:
Homogeneity is correlated with Contrast, r = -0.80
Homogeneity is correlated with Dissimilarity, r = -0.95
GLCM Variance is correlated with Contrast, r= 0.89
GLCM Variance is correlated with Dissimilarity, r= 0.91
GLCM Variance is correlated with Homogeneity, r= -0.83
Entropy is correlated with ASM, r= -0.87
GLCM Mean and Correlation are more independent. For the same image, GLCM Mean shows r< 0.1 with any of the other texture measures demonstrated in this tutorial. GLCM Correlation shows r<0.5 with any other measure. for a review of a lot of feature extraction algorithms look at: Williams et al, 2012, J. of Electronic Imaging, 21(2), 023016 (2012)
glcm <-> haralick "mean" <-> "advanced 1", "variance" <-> "advanced 2", "homogeneity" <-> "simple 4", "contrast"<-> "simple 5", "dissimilarity" <-> "advanced 2", "entropy" <-> "simple 2", "second_moment"<-> "simple 4", "correlation" <-> "simple 3" Furthermore using stats will cover mean and variance while dissimilarity is highly correlated to homogeneity data.

Author(s)

Hanna Meyer

See Also

glcm package

Examples

```
require(glcm)
## example on how to calculate texture with glcm
owd <- getwd()
setwd(tempdir())
data("pacman")
# call glcm wrapper
result <- glcm_texture(pacman,
                       nrasters=1:3,
                       stats=c("mean", "variance", "homogeneity"),
                       parallel = FALSE)

#plot the result:
raster::plot(result[[1]])
setwd(owd)
```

lastool *simple wrapper for some LAStools functions*

Description

simple wrapper for some lastools functions

Usage

```
lastool(tool = "lasinfo", lasFile = NULL, thinGrid = "1.0",
  keepClass = "2", bulge = "1.5", stepSize = "city",
  subSize = "ultra_fine", gridSize = "1.0",
  proj4 = "+proj=utm +zone=32 +datum=WGS84 +units=m +no_defs",
  rscale = "0.01 0.01 0.01", stepoverlap = NULL, cores = "2",
  xoff = 0, yoff = 0, outpath = NULL, pathLastools = NULL,
  verbose = FALSE, cutExtent = NULL, cutSlice = NULL)
```

Arguments

tool	default is lasinfo additionally you may choose lasrepair, lasthin, lasmerge, lasground_new, las2dem, las2txt, lasoverage, lasclip
lasFile	default is NULL path to the laz/las file(s)
thinGrid	default 0.5 meter. Grid stepsize for data thinning
keepClass	default is 2. Default ground class of las/laz conform data
bulge	default is 1.5. 'A parameter to filter spikes it is set to a stepSize/10 and then clamped into the range from 1.0 to 2.0
stepSize	default is 25 meter. lastools key words if city,town,metro,nature,wilderness or experiment with free values
subSize	= "8", default is 8 meter. lastools key words if extra_coarse,coarse,fine,extra_fine,ultra_fine,hyp or experiment with free values
gridSize	resolution of the DTM raster
proj4	default is EPSG 32632, any valid proj4 string that is assumingly the correct one
rscale	rscale
stepoverlap	Spacing of overlap steps aused in lasoverage, default is NULL
cores	number of cores that will be used
xoff	xoff
yoff	yoff
outpath	outpath
pathLastools	character. folder containing the Windows binary files of the lastools
verbose	keep it quiet
cutExtent	NULL
cutSlice	NULL

Author(s)

Chris Reudenbach

Examples

```
## Not run:

require(uavRst)
require(link2GI)
# get a laz file from Mr. Isenburg
utils::download.file(url="http://www.cs.unc.edu/~isenburg/lastools/download/test/s1885565.laz",
                           destfile="test.laz",  quiet = TRUE, mode = "wb")
# convert from laz to las
lastool(tool="las2las","test.laz")


## End(Not run)
```

---

linkAll *convenient function to establish all link2GI links*

Description

brute force search, find and linkl of all link2GI link functions

Usage

```
linkAll(links = NULL, simple = TRUE, linkItems = c("saga", "grass7",
  "otb", "gdal"), sagaArgs = "default", grassArgs = "default",
  otbArgs = "default",
  gdalArgs = c("quiet = TRUE,\n                                 returnPaths = TRUE"))
```

Arguments

links	character. links
simple	logical. true make all
linkItems	character. list of c("saga","grass7","otb","gdal")
sagaArgs	character. full string of sagaArgs
grassArgs	character. grassArgs full string of grassArgs
otbArgs	character. full string of otbArgs
gdalArgs	character. full string of gdalArgs

Note

You may also use the full parameterization of the link2GI package, but you are strungly advaced to use the link2GI functions in a direct way.

Examples

```
## Not run:
# required packages
require(uavRst)
require(link2GI)

# search, find and create the links to all supported  GI software
giLinks<-uavRst::linkAll()


## End(Not run)
```

---

morpho_dem *calculates most important DEM parameters*

Description

calculates most important DEM parameters

Usage

```
morpho_dem(dem, item = c("hillshade", "slope", "aspect", "TRI", "TPI",
  "Roughness", "SLOPE", "ASPECT", "C_GENE", "C_PROF", "C_PLAN", " C_TANG",
  " C_LONG", "C_CROS", "C_MINI", "C_MAXI", "C_TOTA", "C_ROTO", "MTPI"),
  verbose = FALSE, morphoMethod = 6, minScale = 1, maxScale = 8,
  numScale = 2, retRaster = TRUE, giLinks = NULL)
```

Arguments

dem	character filname to GeoTiff containing one channel DEM
item	character list containing the keywords of the DEM parameter to be calculated. Default parameter are c("hillshade", "slope", "aspect", "TRI", "TPI", "Roughness", "SLOPE", "ASPECT", "C_GENE", "C_PROF", "C_PLAN", " C_TANG"," C_LONG", "C_CROS", "C_MINI", "C_MAXI", "C_TOTA", "C_ROTO", "MTPI")
verbose	logical. be quiet
morphoMethod	numeric. saga morphometric method see also: SAGA GIS Help. GDAL parameters see also: gdaldem
minScale	numeric. in scale for multi scale TPI see also: SAGA GIS Help
maxScale	numeric. max scale for multi scale TPI see also: SAGA GIS Help
numScale	numeric. number of scale for multi scale TPI see also: SAGA GIS Help
retRaster	boolean if TRUE a raster stack is returned
giLinks	list. of GI tools cli pathes

Note

please provide a GeoTiff file

Examples

```
## Not run:
##- required packages
require(uavRst)
require(link2GI)
setwd(tempdir())
giLinks<-list()
## check if OTB exists
giLinks$otb <- link2GI::linkOTB()
giLinks$saga <- link2GI::linkSAGA()
giLinks$grass <- link2GI::linkGRASS7(returnPaths = TRUE)
if (giLinks$otb$exist & giLinks$saga$exist & giLinks$grass$exist) {
data("mrbiko")
proj = "+proj=utm +zone=32 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0"
mrbiko <- raster::projectRaster(mrbiko, crs = proj,method = "ngb",res = 20)
raster::writeRaster(mrbiko,"dem.tif",overwrite=TRUE)
r<-morpho_dem(dem="dem.tif",c("hillshade", "slope", "aspect", "TRI", "TPI",
                              "Roughness", "SLOPE", "ASPECT",  "C_GENE", "C_PROF",
                              "C_PLAN", " C_TANG"," C_LONG", "C_CROS"),
                              giLinks= giLinks)
r_st=raster::stack(r)
names(r_st)=names(r)
raster::plot(r_st)
}

## End(Not run)
```

---

mrbiko *DEM data set of Marburg-Biedenkopf*

Description

DEM data set resampled to 20 m resolution

Format

"raster::raster"

---

otbtex_edge *Calculates edges for a given kernel size.*

Description

Calculates edges for a given kernel size. return list of geotiffs containing thelocal statistics for each channel

Usage

```
otbtex_edge(input = NULL, out = "edge", ram = "8192",
  filter = "gradient", filter.touzi.xradius = 1,
  filter.touzi.yradius = 1, channel = NULL, retRaster = FALSE,
  outDir = NULL, verbose = FALSE, giLinks = NULL)
```

Arguments

input	of GeoTiff containing 1 ore more gray value band(s)
out	the output mono band image containing the edge features
ram	reserved memory in MB
filter	the choice of edge detection method (gradient/sobel/touzi)
filter.touzi.xradius	x radius of the Touzi processing neighborhood (if filter==touzi) (default value is 1 pixel)
filter.touzi.yradius	y radius of the Touzi processing neighborhood (if filter==touzi) (default value is 1 pixel)
channel	sequence of bands to be processed
retRaster	boolean if TRUE a raster stack is returned
outDir	output Directory
verbose	switch for system messages default is FALSE
giLinks	list. of GI tools cli pathes

Note

the otb is used for filtering. please provide a GeoTiff file

Author(s)

Chris Reudenbach

Examples

```
## Not run:
##- required packages
require(uavRst)
require(link2GI)
setwd(tempdir())

## check if OTB exists
giLinks <- list()
giLinks$otb <- link2GI::linkOTB()

if (giLinks$otb$exist) {
data("pacman")
pacman<-raster::disaggregate(pacman,10)
```

```
raster::writeRaster(pacman,"pacman.tif",overwrite=TRUE)

##- calculate Sobel edge detection
r<-otbtex_gray(input="pacman.tif",
               filter = "erode",
               structype = "ball",
               structype.ball.xradius = 3,
               structype.ball.yradius = 3 ,
               retRaster = TRUE)

##- visualize all layers
raster::plot(r[[1]])
}

## End(Not run)
```

---

otbtex_gray *Calculates Gray scale morphological operations for a given kernel size.*

Description

Calculates Gray scale morphological operations for a given kernel size. return list of geotiffs containing thelocal statistics for each channel

Usage

```
otbtex_gray(input = NULL, out = "morpho", ram = "8192",
  filter = "dilate", structype = "ball", structype.ball.xradius = 5,
  structype.ball.yradius = 5, channel = NULL, retRaster = FALSE,
  outDir = NULL, verbose = FALSE, giLinks = NULL)
```

Arguments

input	of GeoTiff containing 1 ore more gray value bands
out	the output mono band image containing the edge features
ram	reserved memory in MB
filter	the choice of the morphological operation (dilate/erode/opening/closing) (default value is dilate)
structype	the choice of the structuring element type (ball/cross)
structype.ball.xradius	x the ball structuring element X Radius (only if structype==ball)
structype.ball.yradius	y the ball structuring element Y Radius (only if structype==ball)
channel	sequence of bands to be processed
retRaster	boolean if TRUE a raster stack is returned

outDir	output Directory
verbose	switch for system messages default is FALSE
giLinks	list. of GI tools cli pathes

Note

the otb is used for filtering. please provide a GeoTiff file

Author(s)

Chris Reudenbach

Examples

```
## Not run:
require(uavRst)
require(link2GI)
setwd(tempdir())
## check if OTB exists
giLinks<-list()
giLinks$otb <- link2GI::linkOTB()

if (giLinks$otb$exist) {
data("pacman")
raster::writeRaster(pacman,"pacman.tif",overwrite=TRUE)
r<-otbtex_gray(input="pacman.tif",retRaster = TRUE)

##- visualize all layers
raster::plot(r[[1]])
}

## End(Not run)
```

---

otbtex_hara — *OTB wrapper for Haralick's simple, advanced and higher order texture features.*

Description

OTB wrapper for calculating Haralick's simple, advanced and higher order texture features on every pixel in each channel of the input image

Usage

```
otbtex_hara(x, texture = "all", output_name = "hara",
  path_output = NULL, return_raster = FALSE,
  parameters.xyrad = list(c(1, 1)), parameters.xyoff = list(c(1, 1)),
  parameters.minmax = c(0, 255), parameters.nbbin = 8,
  channel = NULL, verbose = FALSE, giLinks = NULL, ram = "8192")
```

Arguments

x	A Raster* object or a GeoTiff containing one or more gray value bands
texture	type of filter "all" for all, alternative one of "simple" "advanced" "higher"
output_name	string pattern vor individual naming of the output file(s)
path_output	path outut
return_raster	boolean if TRUE a raster stack is returned
parameters.xyrad	list with the x and y radius in pixel indicating the kernel sizes for which the textures are calculated
parameters.xyoff	vector containg the directional offsets. Valid combinations are: list(c(1,1),c(1,0),c(0,1),c(1,-1))
parameters.minmax	minimum/maximum gray value which can occur.
parameters.nbbin	number of gray level bins (classes)
channel	sequence of bands to be processed
verbose	switch for system messages default is FALSE
giLinks	list. of GI tools cli pathes
ram	reserved memory in MB

Details

More information at: texture tutorial Keep in mind that:
Homogeneity is correlated with Contrast, r = -0.80
Homogeneity is correlated with Dissimilarity, r = -0.95
GLCM Variance is correlated with Contrast, r= 0.89
GLCM Variance is correlated with Dissimilarity, r= 0.91
GLCM Variance is correlated with Homogeneity, r= -0.83
Entropy is correlated with ASM, r= -0.87
GLCM Mean and Correlation are more independent. For the same image, GLCM Mean shows r< 0.1 with any of the other texture measures demonstrated in this tutorial. GLCM Correlation shows r<0.5 with any other measure. for a review of a lot of feature extraction algorithms look at: Williams et al, 2012, J. of Electronic Imaging, 21(2), 023016 (2012)
glcm <-> haralick "mean" <-> "advanced 1", "variance" <-> "advanced 2", "homogeneity" <-> "simple 4", "contrast"<-> "simple 5", "dissimilarity" <-> "advanced 2", "entropy" <-> "simple 2", "second_moment"<-> "simple 4", "correlation" <-> "simple 3" Furthermore using stats will cover mean and variance while dissimilarity is highly correlated to homogeneity data.

Author(s)

Chris Reudenbach

References

Haralick, R.M., K. Shanmugam and I. Dinstein. 1973. Textural Features for Image Classification. IEEE Transactions on Systems, Man and Cybernetics. SMC-3(6):610-620.

Orfeo Toolbox Sofware Guide, 2016

"simple":

computes the following 8 local Haralick textures features: Energy, Entropy, Correlation, Inverse Difference Moment, Inertia, Cluster Shade, Cluster Prominence and Haralick Correlation. They are provided in this exact order in the output image. Thus, this application computes the following Haralick textures over a neighborhood with user defined radius.

To improve the speed of computation, a variant of Grey Level Co-occurrence Matrix(GLCM) called Grey Level Co-occurrence Indexed List (GLCIL) is used. Given below is the mathematical explanation on the computation of each textures. Here `g( i,j)` is the frequency of element in the GLCIL whose index is `i,j`. GLCIL stores a pair of frequency of two pixels from the given offset and the cell index `(i,j)` of the pixel in the neighborhood window. Where each element in GLCIL is a pair of pixel index and it's frequency, `g(i,j)` is the frequency value of the pair having index is `i,j`.

Energy
Entropy
Correlation
Inertia (contrast)
Cluster Shade
Cluster Prominence
Haralick's Correlation

"advanced":

computes the following 10 texture features: Mean, Variance, Dissimilarity, Sum Average, Sum Variance, Sum Entropy, Difference of Entropies, Difference of Variances, IC1 and IC2. They are provided in this exact order in the output image. The textures are computed over a sliding window with user defined radius. To improve the speed of computation, a variant of Grey Level Co-occurrence Matrix(GLCM) called Grey Level Co-occurrence Indexed List (GLCIL) is used. Given below is the mathematical explanation on the computation of each textures. Here `g( i,j)` is the frequency of element in the GLCIL whose index is `i,j`. GLCIL stores a pair of frequency of two pixels from the given offset and the cell index `( i,j)` of the pixel in the neighborhood window. (where each element in GLCIL is a pair of pixel index and it's frequency, `g( i,j)` is the frequency value of the pair having index is `i,j`.

Mean
Sum of squares: Variance
Dissimilarity
Sum average
Sum Variance
Sum Entropy
Difference variance
Difference entropy
Information Measures of Correlation IC1
Information Measures of Correlation IC2

"higher":

computes 11 local higher order statistics textures coefficients based on the grey level run-length matrix. It computes the following Haralick textures over a sliding window with user defined radius: (where p(i,j) is the element in cell i,j of a normalized Run Length Matrix (n_r) is the total number of runs and n_p is the total number of pixels):

Short Run Emphasis
Long Run Emphasis
Grey-Level Nonuniformity
Run Length Nonuniformity
Low Grey-Level Run Emphasis
High Grey-Level Run Emphasis
Short Run Low Grey-Level Emphasis
Short Run High Grey-Level Emphasis
Long Run Low Grey-Level Emphasis
Long Run High Grey-Level Emphasis

Examples

```
## Not run:
require(uavRst)
require(link2GI)
## -check if OTB is installed correctly
giLinks <- uavRst::linkAll()
if (giLinks$otb$exist) {
setwd(tempdir())
##- get some typical data as provided by the authority
tmp<-Sys.setlocale('LC_ALL','C')
utils::download.file(url="http://www.ldbv.bayern.de/file/zip/5619/DOP%2040_CIR.zip",
                     destfile="testdata.zip")
unzip("testdata.zip",junkpaths = TRUE,overwrite = TRUE)

# calculate simple Haralick-textures
r<- otbtex_hara(x="4490600_5321400.tif",texture = "simple",return_raster = TRUE)

#plot the results :
##- visualize all layers
raster::plot(r)
tmp<-Sys.setlocale(category = "LC_ALL", locale = "de_DE.-8")
}

## End(Not run)
```

---

otb_stat *Calculates local statistics for a given kernel size*

Description

Calculates local statistics for a given kernel size

Usage

```
otb_stat(input = NULL, out = "localStat", ram = "8192", radius = 3,
  channel = NULL, retRaster = FALSE, outDir = NULL,
  verbose = FALSE, giLinks = NULL)
```

Arguments

input	of GeoTiff containing 1 ore more gray value bands
out	string pattern vor individual naming of the output file(s)
ram	reserved memory in MB
radius	computational window in pixel
channel	sequence of bands to be processed
retRaster	boolean if TRUE a raster stack is returned
outDir	output Directory
verbose	switch for system messages default is FALSE
giLinks	list. of GI tools cli pathes

Note

the otb is used for the calculation of the statistics. Please provide a GeoTiff file

Author(s)

Chris Reudenbach

Examples

```
## Not run:
require(uavRst)
# check if OTB is installed correctly
giLinks <- uavRst::linkAll()
if (giLinks$otb$exist) {
setwd(tempdir())
data("pacman")
raster::writeRaster(pacman,"pacman.tif",overwrite=TRUE)

# calculate statistics
result<- otb_stat(input="pacman.tif",radius=5,retRaster = TRUE)
#plot the results :
raster::plot(result[[1]])
}

## End(Not run)
```

pacman *example raster data set for demonstration usage*

Description

dump of the well know pac man game

Format

"raster::raster"

pc2D_dsm *Create a Digital Terrain Model from UAV generated point clouds by minimum altitude sampling*

Description

Create a Digital Surface Model from a high density point cloud as typically derived by an optical UAV retrieval. It simply samples the maximum or whatever values of a given target grid size and fills the no data holes if so.

Usage

```
pc2D_dsm(laspcFile = NULL, gisdbasePath = NULL, grassVersion = 1,
  searchPath = NULL, sampleMethod = "max", threshold = 20,
  cutExtent = NULL, targetGridSize = 0.25, projFolder = NULL,
  proj4 = "+proj=utm +zone=32 +datum=WGS84 +units=m +no_defs",
  giLinks = NULL, verbose = FALSE)
```

Arguments

laspcFile	character. default is NULL path to the laz/las file(s)
gisdbasePath	character. default is NULL root directory of the project. NOTE the function creates two subfolder named run and output
grassVersion	numeric. version of GRASS as derived by findGRASS() default is 1 (=oldest/only version) please note GRASS version later than 7.4 is not working with r.inlidar
searchPath	path to look for grass
sampleMethod	sampling method for point aggregation
threshold	numeric. percentile threshold
cutExtent	clip area
targetGridSize	numeric. the resolution of the target DTM raster
projFolder	subfolders that will be created/linked for R related GRASS processing

proj4	character. valid proj4 string that should be assumingly the correct one
giLinks	list of link2GI cli pathes, default is NULL
verbose	to be quiet (1)

Author(s)

Chris Reudenbach

Examples

```
## Not run:
require(uavRst)
require(link2GI)

# create and check the links to the GI software
giLinks<-list()
giLinks$grass<-link2GI::linkGRASS7(returnPaths = TRUE)
if (giLinks$grass$exist) {

# proj subfolders
owd <- getwd()
projRootDir<-tempdir()
unlink(paste0(projRootDir,"*"), force = TRUE)
paths<-link2GI::initProj(projRootDir = projRootDir,
                         projFolders = c("data/","data/ref/","output/","run/","las/"),
                         global = TRUE,
                         path_prefix = "path_")
# get some colors
pal = mapview::mapviewPalette("mapviewTopoColors")

# get the data
utils::download.file(url="https://github.com/gisma/gismaData/raw/master/uavRst/data/lidar.las",
destfile="lasdata.las")

# create 2D pointcloud DSM
dsm <- pc2D_dsm(laspcFile = "lasdata.las",
                gisdbasePath = projRootDir,
                sampleMethod = "max",
                targetGridSize = 0.5,
                giLinks = giLinks)
                }
 raster::plot(dsm)
 setwd(owd)

## End(Not run)
```

---

pc2D_dtm *Create a Digital Terrain Model from UAV generated point clouds by minimum altitude sampling (moving window)*

Description

Create a Digital Terrain Model from a high density point cloud as typically derived by an optical UAV retrieval. Due to the poor estimation of ground points a minimum samopling approach is applied. It retrieves on a coarse sampling gridsize the minimum value and interpolates on these samples a surface grid with a higher target resolution. this is a kind of an try and error process and provides fairly good results if the point cloud shows at least some real surface points on a not to coarse grid.

Usage

```
pc2D_dtm(laspcFile = NULL, gisdbasePath = projRootDir,
  grassVersion = 1, searchPath = NULL, tension = 30,
  sampleMethod = "min", cutExtent = NULL, sampleGridSize = 50,
  winRes = c(100, 50, 25), targetGridSize = 0.05,
  splineThresGridSize = 0.5, projFolder = NULL,
  proj4 = "+proj=utm +zone=32 +datum=WGS84 +units=m +no_defs +ellps=WGS84",
  giLinks = giLinks, verbose = FALSE)
```

Arguments

laspcFile	character. default is NULL path to the laz/las file(s)
gisdbasePath	character. default is NULL root directory of the project. NOTE the function creates two subfolder named run and output
grassVersion	numeric. version of GRASS as derived by findGRASS() default is 1 (=oldest/only version) please note GRASS version later than 7.4 is not working with r.inlidar
searchPath	path to look for grass
tension	numeric. tension of spline interpolation.
sampleMethod	character. sampling method of r.in.lidar Statistic to use for raster values Options: n, min, max, range, sum, mean, stddev, variance, coeff_var, median, percentile, skewness, trimmean Default: mean
cutExtent	clip area
sampleGridSize	numeric, resolution extraction raster
winRes	list of moving window resolution for optimize Ground model
targetGridSize	numeric. the resolution of the target DTM raster
splineThresGridSize	numeric. threshold of minimum gridsize tha is used for splininterpolation if the desired resolution is finer a two step approximation is choosen first step spline interpolation using the treshold gridsize second step bilinear resampling to the desired targetGridSize.
projFolder	subfolders that will be created/linked for R related GRASS processing
proj4	character. valid proj4 string that should be assumingly the correct one
giLinks	list of link2GI cli pathes, default is NULL
verbose	to be quiet (1)

Author(s)

Chris Reudenbach, Finn Möller

Examples

```
## Not run:

require(uavRst)
require(link2GI)

# create and check the links to the GI software
giLinks<-list()
giLinks$grass<-link2GI::linkGRASS7(returnPaths = TRUE)
if (giLinks$grass$exist) {

# proj subfolders
projRootDir<-tempdir()
paths<-link2GI::initProj(projRootDir = tempdir(),
                         projFolders = c("data/","data/ref/","output/","run/","las/"),
                         global = TRUE,
                         path_prefix = "path_")
# get the data
utils::download.file(url="https://github.com/gisma/gismaData/raw/master/uavRst/data/lidar.las",
destfile="lasdata.las")

# create 2D point cloud DTM
dtm <- pc2D_dtm(laspcFile = "lasdata.las",
                gisdbasePath = tempdir(),
                tension = 20 ,
                sampleGridSize = 25,
                targetGridSize = 0.5,
                giLinks = giLinks)
 raster::plot(dtm)
}

## End(Not run)
```

---

pc2D_dtm_fw *Create a Digital Terrain Model from UAV generated point clouds by minimum altitude sampling (fix window)*

Description

Create a Digital Terrain Model from a high density point cloud as typically derived by an optical UAV retrieval. Due to the poor estimation of ground points a minimum samopling approach is applied. It retrieves on a coarse sampling gridsize the minimum value and interpolates on these samples a surface grid with a higher target resolution. this is a kind of an try and error process and provides fairly good results if the point cloud shows at least some real surface points on a not to coarse grid.

Usage

```
pc2D_dtm_fw(laspcFile = NULL, grassVersion = 1, searchPath = NULL,
  gisdbasePath = NULL, tension = 20, sampleMethod = "min",
  cutExtent = NULL, sampleGridSize = 25, targetGridSize = 0.25,
  splineThresGridSize = 0.5, projFolder = NULL,
  proj4 = "+proj=utm +zone=32 +datum=WGS84 +units=m +no_defs",
  giLinks = NULL, verbose = FALSE)
```

Arguments

laspcFile	character. default is NULL path to the laz/las file(s)
grassVersion	numeric. version of GRASS as derived by findGRASS() default is 1 (=oldest/only version) please note GRASS version later than 7.4 is not working with r.inlidar
searchPath	path to look for grass
gisdbasePath	character. default is NULL root directory of the project. NOTE the function creates two subfolder named run and output
tension	numeric. tension of spline interpolation.
sampleMethod	character. sampling method of r.in.lidar Statistic to use for raster values Options: n, min, max, range, sum, mean, stddev, variance, coeff_var, median, percentile, skewness, trimmean Default: mean
cutExtent	clip area
sampleGridSize	numeric, resolution extraction raster
targetGridSize	numeric. the resolution of the target DTM raster
splineThresGridSize	numeric. threshold of minimum gridsize tha is used for splininterpolation if the desired resolution is finer a two step approximation is choosen first step spline interpolation using the treshold gridsize second step bilinear resampling to the desired targetGridSize.
projFolder	subfolders that will be created/linked for R related GRASS processing
proj4	character. valid proj4 string that should be assumingly the correct one
giLinks	list of link2GI cli pathes, default is NULL
verbose	to be quiet (1)

Author(s)

Chris Reudenbach

Examples

```
## Not run:
require(uavRst)
require(link2GI)

# create and check the links to the GI software
```

```
giLinks<-list()
giLinks$grass<-link2GI::linkGRASS7(returnPaths = TRUE)
if (giLinks$grass$exist) {

# proj subfolders
projRootDir<-tempdir()
unlink(paste0(projRootDir,"*"), force = TRUE)
paths<-link2GI::initProj(projRootDir = projRootDir,
                         projFolders = c("data/","data/ref/","output/","run/","las/"),
                         global = TRUE,
                         path_prefix = "path_")

# get some colors
pal = mapview::mapviewPalette("mapviewTopoColors")

# get the data
utils::download.file(url="https://github.com/gisma/gismaData/raw/master/uavRst/data/lidar.las",
destfile=paste0("lasdata.las"))

# create 2D point cloud DTM
dtm <- pc2D_dtm_fw(laspcFile = "lasdata.las",
                gisdbasePath = projRootDir,
                tension = 20 ,
                sampleGridSize = 25,
                targetGridSize = 0.5,
                giLinks = giLinks)

mapview::mapview(dtm)
                }


## End(Not run)
```

---

pc3D_dsm *Create a Digital Surface Model from a UAV generated point cloud*

Description

Create a Digital Surface Model from a UAV generated point cloud. Basically returns a DSM. It uses the 'r.in.lidar' function to calculate LiDAR derived raster grids. It creates a `raster*` object.

Usage

```
pc3D_dsm(lasDir = NULL, gisdbasePath = NULL, GRASSlocation = "tmp/",
  grassVersion = 1, searchPath = NULL, projsubFolders = c("data/",
  "output/", "run/", "las/"), gridSize = "0.25",
  grass_lidar_method = "mean", grass_lidar_pth = 90,
  splineNumber = "4", otb_gauss_radius = "0.5",
  smoothFilter = "gauss", dsm_minalt = 0, dsm_maxalt = 4000,
  dsm_area = FALSE, cutExtent = NULL,
```

```
proj4 = "+proj=utm +zone=32 +datum=WGS84 +units=m +no_defs",
gisdbase_exist = FALSE, pathLastools = NULL, giLinks = NULL,
MP = "~", verbose = FALSE)
```

Arguments

lasDir	character. default is NULL path to the laz/las file(s)
gisdbasePath	character. gisdbase will be linked or created depending on gisdbase_exist
GRASSlocation	character. location will be linked or created depending on gisdbase_exist
grassVersion	numeric. version of GRASS as derived by findGRASS() default is 1 (=oldest/only version) please note GRASS version later than 7.4 is not working with r.inlidar
searchPath	path to look for grass
projsubFolders	character. subfolders that will be created/linked for R related GRASS processing
gridSize	numeric. resolution for raster operations
grass_lidar_method	character. statistical method to sample the Lidar data, see also r.in.lidar help.
grass_lidar_pth	numeric. grass lidar aggregation column percentile, see also r.in.lidar help.
splineNumber	numeric. default is 4 number ob spline iterations
otb_gauss_radius	numeric. default is 0.5 radius of otb smoothing filter in meter
smoothFilter	character. default is gauss alternatives are spline or for no smoothing at all no
dsm_minalt	numeric. default is 0, minimum DTM altitude accepted
dsm_maxalt	numeric. dsm maximum altitude
dsm_area	numeric. default FALSE generate polygon of valid DSM data
cutExtent	numerical. clip area c(mix,miny,maxx,maxy)
proj4	character. default is EPSG 32632 any valid proj4 string that is assumingly the correct one
gisdbase_exist	logical. default is FALSE switch if gisdbase is created or linked only
pathLastools	character. folder containing the Windows binary files of the lastools
giLinks	list. of GI tools cli paths as generated by a full call of link2GI
MP	character mounting point / drive letter default is "~"
verbose	logical. to be quiet FALSE

Note

For using 'PDAL'(since GRASS7.4.x) you have to install the PDAL binaries and python bindings if not bundled with GRASS. running Linux `sudo apt-get install libpdal-base5 libpdal-dev libpdal-plugin-pytho`

Author(s)

Chris Reudenbach

See Also

r.in.lidar help

Examples

```
## Not run:
require(uavRst)
require(link2GI)

# create and check the links to the GI software
giLinks<-uavRst::linkAll()
(giLinks$saga$exist & giLinks$otb$exist & giLinks$grass$exist)
{
# proj subfolders
projRootDir<-tempdir()
setwd(projRootDir)
unlink(paste0(projRootDir,"*"), force = TRUE)

projsubFolders<-c("data/","data/ref/","output/","run/","las/")
paths<-link2GI::initProj(projRootDir = projRootDir,
                         projFolders = projsubFolders,
                         global = TRUE,
                         path_prefix = "path_")
# get some colors
pal = mapview::mapviewPalette("mapviewTopoColors")

# get the data
utils::download.file(url="https://github.com/gisma/gismaData/raw/master/uavRst/data/lidar.las",
                     destfile="lasdata.las")

# create a DSM  based on a uav point cloud
pc3DSM<-pc3D_dsm(lasDir =  "lasdata.las",
        gisdbasePath = projRootDir,
        projsubFolders = projsubFolders,
        gridSize = "0.5",
        giLinks = giLinks)
mapview::mapview(pc3DSM[[1]])
}
##+
## End(Not run)
```

---

pc3D_dtm *create a Digital Terrain Model from preclassified point cloud data*

Description

Create a Digital Terrain Model from a high density point cloud as typically derived by an optical UAV retrieval.

Usage

```
pc3D_dtm(lasDir = NULL, gisdbasePath = NULL, thinGrid = "0.5",
  keepClass = "2", bulge = "1.5", splineNumber = "4",
  stepSize = "city", subSize = "ultra_fine", gridSize = "0.25",
  dtmarea = FALSE, cutExtent = NULL, projsubFolders = c("data/",
  "output/", "run/", "las/"),
  proj4 = "+proj=utm +zone=32 +datum=WGS84 +units=m +no_defs",
  cores = "3", pathLastools = NULL, giLinks = NULL, MP = "~",
  verbose = FALSE)
```

Arguments

lasDir	character. default is NULL path to the laz/las file(s)
gisdbasePath	character. default is NULL root directory of the project. NOTE the function creates two subfolder named run and output
thinGrid	numerical. default 0.5 meter. Grid stepsize for data thinning
keepClass	numerical. default is 2. Default ground class of las/laz conform data
bulge	numerical. default is 1.5. 'A parameter to filter spikes it is set to a stepSize/10 and then clamped into the range from 1.0 to 2.0
splineNumber	numerical. default is 3, Maximum number of spline iterations highly suggested to take odd numbers. As higher as more detailed (spurious).
stepSize	character. default is 25 meter. lastools key words if city,town,metro,nature,wilderness or experiment with free values
subSize	= character. "8", default is 8 meter. lastools key words if extra_coarse,coarse,fine,extra_fine,ultra or experiment with free values
gridSize	numerical. resolution of the DTM raster
dtmarea	logical. default FALSE generate polygon of valid DTM data
cutExtent	object of typ extent deteerming the clip area
projsubFolders	list of character contaiing subfolders that will be created/linked for R related GRASS processing
proj4	default is EPSG 32632, any valid proj4 string that is assumingly the correct one
cores	numerical. number of cores that will be used
pathLastools	character. folder containing the Windows binary files of the lastools
giLinks	list of GI tools cli pathes, default is NULL
MP	character mounting point / drive letter default is "~"
verbose	logical. to be quiet (1)

Author(s)

Chris Reudenbach

Examples

```
## Not run:
require(uavRst)
require(link2GI)

# create and check the links to the GI software
giLinks<-list()
giLinks$saga<-link2GI::linkSAGA()
if (giLinks$saga$exist) {
# proj subfolders
projRootDir<-tempdir()
unlink(paste0(projRootDir,"*"), force = TRUE)
projsubFolders<-c("data/","data/ref/","output/","run/","las/")
paths<-link2GI::initProj(projRootDir = projRootDir,
                         projFolders = projsubFolders,
                         global = TRUE,
                         path_prefix = "path_")
setwd(paste0(projRootDir,"run"))
# get some colors
pal = mapview::mapviewPalette("mapviewTopoColors")

# get the data
utils::download.file(url="https://github.com/gisma/gismaData/raw/master/uavRst/data/lidar.las",
                     destfile="lasdata.las")

# create a DSM  based on a uav point cloud
pc3DTM <- pc3D_dtm(lasDir =  "lasdata.las",
                       gisdbasePath = projRootDir,
                       projsubFolders = projsubFolders,
                       thinGrid = 1.,
                       splineNumber = 5 ,
                       gridSize = 0.5,
                       giLinks = giLinks)
mapview::mapview(pc3DTM[[1]])
}

## End(Not run)
```

`poly_metrics` *calculate morphometric features of polygons.*

Description

calculate morphometric features of polygons. Calculate some crown related metrics, returns the metrics as a spatialpointdataframe/spatialpolygondataframe

Usage

```
poly_metrics(crownarea, funNames = c("length", "elongation",
```

```
  "eccentricityboundingbox", "solidity", "eccentricityeigen", "calliper",
  "rectangularity", "circularityharalick", "convexity"))
```

Arguments

crownarea	sp* spatialpolygon object
funNames	character. names of morphometrics to be calculated available are ("length","elongation","eccentricityboun "calliper","rectangularity","circularityharalick","convexity")

See Also

Momocs Paul Rosin

Examples

```
## Not run:
# required packages
require(uavRst)
require(link2GI)
require(mapview)

# project folder
projRootDir<-tempdir()

# create subfolders please mind that the pathes are exported as global variables
paths<-link2GI::initProj(projRootDir = projRootDir,
                         projFolders = c("data/","data/ref/","output/","run/","las/"),
                         global = TRUE,
                         path_prefix = "path_")
# overide trailing backslash issue
 path_run<-ifelse(Sys.info()["sysname"]=="Windows", sub("/$", "",path_run),path_run)
 setwd(path_run)

# get the rgb image, chm and training data
utils::download.file("https://github.com/gisma/gismaData/raw/master/uavRst/data/tutorial_data.zip",
                     paste0(path_run,"tutorial_data.zip"))
unzip(zipfile = paste0(path_run,"tutorial_data.zip"), exdir = path_run)
crown<-raster::shapefile(paste0(path_run,"rgb_3-3_train2.shp"))
## calculate polygon morpho metrics
polymetric <- poly_metrics(crown)

# visualize it
mapview::mapview(polymetric)
##+
## End(Not run)
```

poly_stat *Calculate descriptive statistics of raster as segemented by polygons*

Description

calculate statitiscs of polygon based raster extraction. Returns a spatialpolygon dataframe containing decriptive statistics

Usage

```
poly_stat(x = NULL, spdf = NULL, count = 1, min = 1, max = 1,
  sum = 1, range = 1, mean = 1, var = 1, stddev = 1,
  quantile = 10, parallel = 1, giLinks = NULL)
```

Arguments

x	list of spatial Raster* object(s)
spdf	spatial point dataframe
count	0 1 switch
min	0 1 switch
max	0 1 switch
sum	0 1 switch
range	0 1 switch
mean	0 1 switch
var	0 1 switch
stddev	0 1 switch
quantile	number of quantile
parallel	run it parallel default is 1
giLinks	list of GI tools cli pathes, default is NULL

Author(s)

Chris Reudenbach

Examples

```
## Not run:
# required packages
require(uavRst)
require(link2GI)

# create and check the links to the GI software
giLinks<-uavRst::linkAll()
if (giLinks$saga$exist & giLinks$otb$exist & giLinks$grass$exist) {
```

```
# project folder
projRootDir<-tempdir()

# create subfolders please mind that the pathes are exported as global variables
paths<-link2GI::initProj(projRootDir = projRootDir, projFolders = c("run/"),
                         global = TRUE,
                         path_prefix = "path_")
# overide trailing backslash issue
 path_run<-ifelse(Sys.info()["sysname"]=="Windows", sub("/$", "",path_run),path_run)

# get the rgb image, chm and training data
url <- "https://github.com/gisma/gismaData/raw/master/uavRst/data/tutorial_data.zip"
utils::download.file(url, paste0(path_run,"tutorial_data.zip"))
unzip(zipfile = paste0(path_run,"tutorial_data.zip"), exdir = path_run)

# convert tif to SAGA
gdalUtils::gdal_translate(paste0(path_run,"rgb_3-3_train1.tif"),
                          paste0(path_run,"rgb_3-3_train1.sdat"),
                          overwrite = TRUE,
                          b = 1,
                          of = 'SAGA',
                          verbose = FALSE)

polyStat <- poly_stat("rgb_3-3_train1",
                      spdf = "rgb_3-3_train1.shp",
                      giLinks=giLinks)

mapview::mapview(polyStat)
}
##+
## End(Not run)
```

`predict_rgb` *classify images using raster predict*

Description

classify images using raster predict

Usage

```
predict_rgb(imageFiles = NULL, model = NULL, inPrefix = "index_",
  outPrefix = "classified_", bandNames = NULL)
```

Arguments

`imageFiles`	raster*. imagestack for classification purposes must contain the required bands as needed by the model.

model	model. classification model
inPrefix	character. in frefix string
outPrefix	character. out prefix string
bandNames	character. band names

Examples

```
## Not run:
##- required packages
require(uavRst)
require(link2GI)

##- project folder
projRootDir<-tempdir()

##-create subfolders pls notice the pathes are exported as global variables
paths<-link2GI::initProj(projRootDir = projRootDir,
                         projFolders = c("data/","data/ref/","output/","run/","las/"),
                         global = TRUE,
                         path_prefix = "path_")
##- overide trailing backslash issue
path_run<-ifelse(Sys.info()["sysname"]=="Windows", sub("/$", "",path_run),path_run)
setwd(path_run)
unlink(paste0(path_run,"*"), force = TRUE)

##- get the tutorial data
utils::download.file("https://github.com/gisma/gismaData/raw/master/uavRst/data/ffs.zip",
paste0(path_run,"ffs.zip"))
unzip(zipfile =  paste0(path_run,"ffs.zip"), exdir = path_run)

##- assign tutorial data
imageFile <- paste0(path_run,"predict.tif")
load(paste0(path_run,"tutorialbandNames.RData"))
tutorialModel<-readRDS(file = paste0(path_run,"tutorialmodel.rds"))

##- start the  prediction taking the non optimized model
##- please note the output is saved in the subdirectory path_output
predict_rgb(imageFiles=imageFile,
            model = tutorialModel[[1]],
            bandNames = bandNames)

##- visualise the classification
raster::plot(raster::raster(paste0(path_output,"classified_predict.tif")))
##+
## End(Not run)
```

---

rgb *RGB Orthoimagefrom an arbitray MOF plot*

Description

Example data set containing a RGB Orthoimage of a small plot sampled in the Maburg University Forest (MOF). The resolution is 10 cm. ETRS89 UTM32

Format

`"raster::raster"`

`rgb_indices` *RGB indices*

Description

This function calculates various spectral indices from a RGB. It returns at least red green and blue as splitted channels in a stack. Additionally you can choose RGB indices. `Raster*` object.

Usage

```
rgb_indices(red, green, blue, rgbi = c("VVI", "VARI", "NDTI", "RI",
  "SCI", "BI", "SI", "HI", "TGI", "GLI", "NGRDI", "GRVI", "GLAI", "HUE",
  "CI", "SAT", "SHP"))
```

Arguments

`red`	a single `Raster` or `RasterBrick` band.
`green`	a single `Raster` or `RasterBrick` band.
`blue`	a single `Raster` or `RasterBrick` band.
`rgbi`	the implemented RGB indices currently • `BI` sqrt((R**2+G**2+B*2)/3 Brightness Index • `SCI` (R-G)/(R+G) Soil Colour Index • `GLI` (2*g - r - b)/(2*g + r + b) Green leaf index Vis Louhaichi et al. (2001) • `HI` (2*R-G-B)/(G-B) Primary colours Hue Index • `NDTI` (R-G)/(R+G) Normalized difference turbidity index Water • `NGRDI` (G-R)/(G+R) Normalized green red difference index (sometimes GRVI) Tucker (1979) • `RI` R**2/(B*G**3) Redness Index • `SI` (R-B)/(R+B) Spectral Slope Saturation Index

- `TGI` -0.5[190(R670-R550)-120(R670 - R480)] The triangular greenness index (TGI) estimates chlorophyll concentration in leaves and canopies
- `VARI` (green-red)/(green+red-blue). A Visible Atmospherically Resistant Index (VARI)
- `VVI` (1-(r-30)/(r+30))*(1-(g-50)/(g+50))*(1-(b-1)/(b+1))
- `GLAI` (25 * (green - red) / (green + red - blue) + 1.25)
- `GRVI` (green-red)/(green+red) Green-Red Vegetation Index
- `CI` (red - blue) / red Coloration Index
- `HUE` atan(2 * (red - green - blue) / 30.5 * (green - blue)) Overall Hue Index
- `SAT` (max(red,green,blue) - min(red,green,blue)) / max(red,green,blue) Overall Saturation Index
- `SHP` 2 * (red - green - blue) / (green - blue) Shape index

References

Planetary Habitability Laboratory (2015): Visible Vegetation Index (VVI). Available online via VVI.
Lacaux, J. P., Tourre, Y. M., Vignolles, C., Ndione, J. A., and Lafaye, M.: Classification of ponds from high-spatial resolution remote sensing: Application to Rift Valley Fever epidemics in Senegal, Remote Sens. Environ., 106, 66-74, 2007.(NDTI))
Gitelson, A., et al.: Vegetation and Soil Lines in Visible Spectral Space: A Concept and Technique for Remote Estimation of Vegetation Fraction. International Journal of Remote Sensing 23 (2002): 2537-2562. (VARI)
MADEIRA, J., BEDIDI, A., CERVELLE, B., POUGET, M. and FLAY, N., 1997, Visible spectrometric indices of hematite (Hm) and goethite (Gt) content in lateritic soils: 5490 N. Levin et al. the application of a Thematic Mapper (TM) image for soil-mapping in Brasilia, Brazil. International Journal of Remote Sensing, 18, pp. 2835-2852.
MATHIEU, R., POUGET, M., CERVELLE, B. and ESCADAFAL, R., 1998, Relationships between satellite-based radiometric indices simulated using laboratory reflectance data and typic soil colour of an arid environment. Remote Sensing of Environment, 66, pp. 17-28.
Louhaichi, M., Borman, M.M., Johnson, D.E., 2001. Spatially located platform and aerial photography for documentation of grazing impacts on wheat. Geocarto International 16, 65-70.
Tucker, C.J., 1979. Red and photographic infrared linear combinations for monitoring vegetation. Remote Sensing of Environment 8, 127-150.
GRVI Green-Red Vegetation Index Remote Sensing 2010, 2, 2369-2387; doi:10.3390/rs2102369
CI IDB Coloration
HUE Index IDB Hue
Saturation Index IDB Saturation
Shape Index IDB Shape

See Also

For a comprehensive overview of remote sensing indices have a look at: A database for remote sensing indices
Approximatly wavelength ranges for overlapping digital camera bands are:

- red 580-670 nm,
- green 480-610 nm,
- blue 400-520 nm

Hunt et al., 2005. However check the manual of your camera.

Examples

```
## ## ##
##- setup environment
require(uavRst)
data(rgb)

##- visualize the image
raster::plotRGB(rgb)

##- calculate the indices
rgbI<-rgb_indices(red   = rgb[[1]],
                  green = rgb[[2]],
                  blue  = rgb[[3]],
                  rgbi = c("NDTI","VARI","TGI"))

##- visualize the indices
raster::plot(rgbI)
##+
```

---

r_in_lidar *wraps the r.in.lidar tool*

Description

simple wrapper for 'r.in.lidar' to calculate LiDAR derived raster grids. It creates a `raster*` object.

Usage

```
r_in_lidar(input = NULL, output = NULL, file = NULL, method = NULL,
  type = NULL, base_raster = NULL, zrange = NULL, zscale = NULL,
  intensity_range = NULL, intensity_scale = NULL, percent = NULL,
  pth = NULL, trim = NULL, resolution = NULL, return_filter = NULL,
  class_filter = NULL, flags = c("e", "n", "overwrite", "o"))
```

Arguments

input	input
output	output
file	file
method	method
type	type
base_raster	base_raster
zrange	zrange
zscale	zscale
intensity_range	intensity_range
intensity_scale	intensity_scale
percent	percent
pth	pth
trim	trim
resolution	resolution
return_filter	return_filter
class_filter	class filter
flags	flags
r.in.lidar	[-penosgijdv]

Author(s)

Chris Reudenbach

See Also

r.in.lidar help

Examples

```
## Not run:
require(uavRst)
##- Straightforward example to generate a DTM
    based on the class 2 minimum returns of a LiDAR file

##- set up environment
path<-tempdir()
setwd(path)

##- get a laz file from Mr. Isenburg
url="http://www.cs.unc.edu/~isenburg/lastools/download/test/s1885565.laz"
utils::download.file(url=url,
                     destfile="test.laz",
```

```
                    quiet = TRUE,
                    mode = "wb")
##- convert it from laz to las (obligatory format for using r.in.lidar)
lastool(tool="laz2las",paste0(path,"/test.laz"))

##- extract extent for setting up GRASS region
ext<-lastool(lasFile =  paste0(path,"/test.las"))

##- set up GRASS
proj4 = "+proj=utm +zone=32 +datum=WGS84 +units=m +no_defs"
result<-link2GI::linkGRASS7(spatial_params = c(ext[2],ext[1],ext[4],ext[3],proj4),
                            resolution = 0.5)

##- use the r.in.lidar tool to generate a pseudo surface model
r_in_lidar(input = paste0(getwd(),"/test.las"),
           output = "dem",
           method = "min",
           resolution = 30,
           class_filter = 2)

##- visualize it
raster::plot(raster::raster(rgrass7::readRAST("dem")))


## End(Not run)
```

`treepos_FT` *'ForestTools' tree top detection*

Description

Implements the variable window filter algorithm (Popescu & Wynne, 2004) for detecting treetops from a canopy height model. Andrew Plowright: R package ForestTools

Usage

```
treepos_FT(chm = NULL, winFun = function(x) {     0.5 * ((x^2) * 0.009
  + 2.51) }, minTreeAlt = 2, maxCrownArea = maxCrownArea,
  verbose = TRUE)
```

Arguments

`chm`	Canopy height model in `raster`, `lasmetrics`, `matrix` or object of `class LAS`. Should be the same that was used to create the input for `treepos`.
`winFun`	function. The function that determines the size of the window at any given location on the canopy. It should take the value of a given CHM pixel as its only argument, and return the desired radius of the circular search window when centered on that pixel.

minTreeAlt	Height threshold (m) below a pixel cannot be a local maximum. Local maxima values are used to define tree tops.
maxCrownArea	numeric. A single value of the maximum individual tree crown radius expected.
verbose	quiet (1) height of treepos.

Examples

```
##- required packages
 require(uavRst)

 data(chm_seg)

##- call ForestTools treepos function
 tposFT <- treepos_FT(chm = chm_seg[[1]],
            minTreeAlt = 10,
            maxCrownArea = 150)

##- visualize it
# mapview::mapview(tposFT)
raster::plot(tposFT)
```

---

treepos_GWS *Find potential tree positions using a canopy height model*

Description

Find potential tree positions using a canopy height model by using an iterative watershed algorithm. Basically returns a vector data sets with the tree crown geometries and a bunch of corresponding indices.

Usage

```
treepos_GWS(chm = NULL, minTreeAlt = 10, minTreeAltParam = "chmQ20",
  minCrownArea = 3, maxCrownArea = 150, join = 1, thresh = 0.1,
  split = TRUE, cores = parallel::detectCores(), giLinks = NULL)
```

Arguments

chm	raster* canopy height model
minTreeAlt	numeric. minimum height of trees to be integrated in the analysis
minTreeAltParam	character. code for the percentile that is used as tree height treshold. It is build using the key letters chmQ and adding the percentile i.e. "10". Default is chmQ20
minCrownArea	numeric. minimum area in square meter (if you use projected data) of the projected tree crowns
maxCrownArea	numeric. maximum area in square meter (if you use projected data) of the projected tree crowns

join	numeric. Join Segments based on Threshold Value, 0=no join, 1=treepos_GWS2saddle diff, 2=treepos_GWS2treepos diff. see also SAGA GIS Help
thresh	numeric. Specify a threshold value as minimum difference between neighboured segments in meter. see also SAGA GIS Help
split	split polygons default is TRUE
cores	number of cores to be used
giLinks	list. of GI tools cli paths

Author(s)

Chris Reudenbach

Examples

```
## Not run:

# required packages
require(uavRst)
require(link2GI)

# create and check the links to the GI software
giLinks<-uavRst::linkAll()
if (giLinks$saga$exist & giLinks$otb$exist & giLinks$grass$exist) {

# project folder
projRootDir<-tempdir()

# create subfolders please mind that the pathes are exported as global variables
paths<-link2GI::initProj(projRootDir = projRootDir,
                         projFolders = c("data/","data/ref/","output/","run/","las/"),
                         global = TRUE,
                         path_prefix = "path_")

 data(chm_seg)

# calculate treepos using uavRst generic approach
 tPos <- uavRst::treepos_GWS(chm = chm_seg[[1]],
                         minTreeAlt = 2,
                         maxCrownArea = 150,
                         join = 1,
                         thresh = 0.35,
                         split=TRUE,
                         cores=1,
                         giLinks = giLinks )
}
##+
## End(Not run)
```

`treepos_lidR` *tree top detection based on local maxima filters as provided by 'lidR'*

Description

Tree top detection based on local maxima filters. There are two types of filter. The first, called for gridded objects, works on images with a matrix-based algorithm. And the second one, called for point clouds, works at the point cloud level without any rasterization. Jean-Romain Roussel and David Auty: R package lidR

Usage

```
treepos_lidR(chm = NULL, movingWin = 7, minTreeAlt = 2)
```

Arguments

`chm`	Canopy height model in `raster`, `lasmetrics`, `matrix` or object of `class LAS`. Should be the same that was used to create the input for `treepos`.
`movingWin`	Size (in pixels) of the moving window to detect local maxima.
`minTreeAlt`	Height threshold (m) below a pixel cannot be a local maximum. Local maxima values are used to define tree tops.

Examples

```
## Not run:
require(uavRst)
## required packages
require(uavRst)

 data(chm_seg)

## find trees
tPoslidR <- treepos_lidR(chm = chm_seg[[1]],
                   movingWin = 3,
                   minTreeAlt = 15)
## visualisation
# mapview::mapview(tPoslidR)
raster::plot(tPoslidR)

## End(Not run)
```

treepos_RL — *'rLiDAR' based tree detection of a LiDAR-derived Canopy Height Model (CHM)*

Description

Detects and computes the location and height of individual trees within the LiDAR-derived Canopy Height Model (CHM). The algorithm implemented in this function is local maximum with a fixed window size. Carlos A. Silva et all.: R package rLiDAR

Usage

```
treepos_RL(chm = NULL, movingWin = 7, minTreeAlt = 2)
```

Arguments

chm	Canopy height model in raster or SpatialGridDataFrame file format. Should be the same that was used to create the input for treepos.
movingWin	Size (in pixels) of the moving window to detect local maxima.
minTreeAlt	Height threshold (m) below a pixel cannot be a local maximum. Local maxima values are used to define tree tops.

Examples

```
## required packages
require(uavRst)

## load data
data(chm_seg)

## find trees
tPosRL <- treepos_RL(chm = chm_seg[[1]],
                     movingWin = 3,
                     minTreeAlt = 10)
## visualisation
raster::plot(tPosRL)
```

trp_seg — *Optional tree position raster map*

Description

Example data set containing optional treepoint positions sampled in the Maburg University Forest (MOF). The resolution is 10 cm. ETRS89 UTM32

Format

"raster::raster"

---

uavRst — *Unmanned Aerial Vehicle Remote Sensing Tools - some cool tools to manipulate and analyze UAV derived RGB ortho imagery and point clouds.*

Description

In general the uavRst remote sensing toolbox tries to support the use of UAV derived imagery and pointclouds as a cheap and easy to use alternative/complement to LiDAR data. Howeverit is far from being mature.
uavRst provides functionality to analyze poor quality RGB images as taken by low budget ready to fly uavs. This includes preconfigured machine learning based classification workflows, comprehensive texture analysis and segmentation algorithms as well as forest relevant calculations of metrics and measures on the derived products.

Details

Unmanned Aerial Vehicle Remote Sensing Tools

Note

For most of the functions you need a bunch of third party software. The most comfortable way to fulfill these requirements is to install 'QGIS', 'GRASS'- and 'SAGA-GIS' following the excellent RQGIS. For most of the LiDAR related operations the great R package lidR is used.

However for some of the basic point cloud related operations you will need to install the 'LAStool' software, that can be downloaded here here and is provided by rapidlasso. Please download it and unzip it as usual. For Windows systems it is by default expected that you put it at C:/LASTools, running Linux at ~/apps/LASTools. For running LAStools tools under Linux you first need to install wine.

All of the mentioned software packages have to be correctly installed. Most of it tested under Windows and Linux and should run... The most easiest way to obtain a fairly good runtime enviroment is to setup Linux as a dual boot system or in a VB. You will find some tutorials and examples at the uavRst Wiki. Please feel free to participate.

Author(s)

Hanna Meyer, Thomas Nauss,Florian Detsch, Lars Opgenoorth, Chris Reudenbach, Environmental Informatics Marburg

Maintainer: Chris Reudenbach <reudenbach@uni-marburg.de>

xyz2tif *Read and Convert xyz DEM/DSM Data as typically provided by the Authorities.*

Description

Read xyz data and generate a Raster* object.

Usage

```
xyz2tif(xyzFN = NULL, epsgCode = "25832")
```

Arguments

xyzFN	ASCII tect file with xyz values
epsgCode	"25832"

Examples

```
## Not run:
##- libraries
require(uavRst)
owd <- getwd()
setwd(tempdir())
##- get typical xyz DEM data in this case from the Bavarian authority
utils::download.file("http://www.ldbv.bayern.de/file/zip/10430/DGM_1_ascii.zip",
                     "testdata.zip")
file<- unzip("testdata.zip",list = TRUE)$Name[2]
unzip("testdata.zip",files = file,  overwrite = TRUE)
##- show structure
head(read.table(file))
##- create tiff file
##- NOTE  for creating a geotiff you have to provide the correct EPSG code from the meta data
xyz2tif(file,epsgCode = "31468")

##- visualize it
raster::plot(raster::raster(file))
setwd(owd)

## End(Not run)
```

Index