Package 'TreeLS'

March 13, 2019

Type Package

Title Terrestrial Point Cloud Processing of Forest Data

Version 1.0

Description Algorithms for tree detection, noise removal, stem modelling, 3D visualization and manipulation of terrestrial 'LiDAR' (but not only) point clouds, currently focusing on high performance applications for forest inventory - being fully compatible with the 'LAS' infrastructure provided by 'lidR'. For in depth descriptions of the stem classification and segmentation algorithms check out Conto et al. (2017) <doi:10.1016/j.compag.2017.10.019>.

URL https://github.com/tiagodc/TreeLS

Depends R (>= 3.3.0), data.table (>= 1.12.0), magrittr (>= 1.5), lidR (>= 2.0.0)

Imports rgl (>= 0.99.0), raster (>= 2.8.19)

License GPL-3

Encoding UTF-8

LazyData true

LinkingTo Rcpp,BH,RcppEigen

RoxygenNote 6.1.0

NeedsCompilation yes

Author Tiago de Conto [aut, cre]

Maintainer Tiago de Conto <ti@forlidar.com.br>

Repository CRAN

Date/Publication 2019-03-13 15:43:25 UTC

R topics documented:

`gpsTimeFilter` *Filter points based on gpstime*

Description

This is a simple wrapper to `lasfilter` that takes as inputs proportional values instead of absolute time stamp values for filtering a point cloud object based on the gpstime field. This function is particularly useful to check narrow intervals of point cloud frames from mobile scanning data.

Usage

```
gpsTimeFilter(las, from = 0, to = 1)
```

Arguments

`las`	`LAS` object.
`from, to`	`numeric` - between 0 and 1 - gpstime quantile limits to filter by

Value

`LAS` object.

Examples

```
file = system.file("extdata", "model_boles.laz", package="TreeLS")
tls = readTLS(file)

### color points according to its chronological time stamp
plot(tls, color='gpstime')

### keep points registered in the 70% to 95% time interval
tls = gpsTimeFilter(tls, .7, .95)
plot(tls, color='gpstime')
```

map.hough *Tree mapping algorithm: Hough Transform*

Description

This function is meant to be used inside `treeMap`. It applies an adapted version of the Hough Transform for circle search. Mode details are given in the sections below.

Usage

```
map.hough(hmin = 1, hmax = 3, hstep = 0.5, pixel_size = 0.025,
  max_radius = 0.25, min_density = 0.1, min_votes = 3)
```

Arguments

`hmin, hmax`	`numeric` - height thresholds within the point cloud in which circle search will be performed.
`hstep`	`numeric` - height interval to perform circle search.
`pixel_size`	`numeric` - pixel side length to discretize the point cloud layers while performing the Hough Transform circle search.
`max_radius`	`numeric` - approximately the largest stem cross section radius expected in the point cloud.
`min_density`	`numeric` - between 0 and 1 - minimum point density within a pixel evaluated on the Hough Transform - i.e. only *dense* point clousters will undergo circle search.
`min_votes`	`integer` - Hough Transform parameter - minimum number of circle intersections over a pixel to assign it as a circle center candidate.

`LAS@data` Special Fields

Each point in the `LAS` object output represents a pixel center that is *possibly* also a stem cross-section center.

The variables describing each point in the output are:

- `Intensity`: number of votes received by that point
- `PointSourceID`: unique stem segment ID (among all trees)
- `Keypoint_flag`: if `TRUE`, the point is the most likely circle center of its stem segment (`PointSourceID`)
- `Radii`: approximate radius estimated by that point - always a multiple of the `pixel_size`
- `TreeID`: unique tree ID of the point
- `TreePosition`: if `TRUE`, the point represents its tree's approximate coordinate

Adapted Hough Transform

The Hough Transform circle search algorithm used in TreeLS applies a constrained circle search on discretized point cloud layers. Tree-wise, the circle search is recursive, in which the search for circle parameters of a stem section is constrained to the *feature space* of the stem section underneath it. Initial estimates of the stem's *feature space* are performed on a *baselise* stem segment - i.e. a low height interval where a tree's bole is expected to be clearly visible in the point cloud. The algorithm is described in detail by Conto et al. (2017).

This adapted version of the algorithm is very robust against outliers, but not against forked or leaning stems.

Tree Selection

An initial tree filter is used to select *probable* trees in the input point cloud. Parallel stacked layers, each one as thick as `hstep`, undergo the circle search within the `hmin`/`hmax` limits. On every layer, pixels above the `min_votes` criterion are clustered, forming *probability zones*. *Probability zones* vertically aligned on at least 3/4 of the stacked layers are assigned as *tree occurrence regions* and exported in the output map.

References

Conto, T. ; Olofsson, K. ; Gorgens, E. B. ; Rodriguez, L. C. E. ; Almeida, G. Performance of stem denoising and stem modelling algorithms on single tree point clouds from terrestrial laser scanning. Computers and Electronics in Agriculture, v. 143, p. 165-176, 2017.

Examples

```
file = system.file("extdata", "model_boles.laz", package="TreeLS")
tls = readTLS(file)
plot(tls)

## build a 3D map of tree occurrences
map = treeMap(tls)
plot(map, color='Radii')

## get a 2D representation of the tree map
xymap = treePositions(map)
head(xymap)
```

---

randomize *Point sampling algorithm: random sample*

Description

This function is meant to be used inside `tlsSample`. It selects points randomly, returning a fraction of the input point cloud.

Usage

```
randomize(p = 0.5)
```

Arguments

p	numeric - between 0 and 1 - proportion of points to keep.

Examples

```
file = system.file("extdata", "pine.laz", package="TreeLS")
tls = readTLS(file)
print(tls)

### thin point cloud - note the point count
tls = tlsSample(tls, randomize(0.33))
print(tls)
```

---

readTLS *Import a point cloud file into a LAS object*

Description

Wrapper to read point clouds straight to LAS objects suitable for TLS applications. Reads *las* or *laz* files with readLAS and alters the header defaults. Other file formats are (or try to be) read using read.table.

Usage

```
readTLS(file, colNames = NULL, ...)
```

Arguments

file	file path.
colNames	optional - character vector. Only used for table-like files. It states the column names - if not set, only the 3 first columns will be converted to XYZ.
...	further arguments passed to either readLAS or read.table.

Value

LAS object.

Examples

```
file = system.file("extdata", "pine.laz", package="TreeLS")
tls = readTLS(file)
summary(tls)
```

setTLS *Reset or create a* LAS *object depending on the input's type*

Description

Reset the input's header if it is a LAS object, or generate a new LAS from a table-like input. For more information, checkout the lidR::LAS description page.

Usage

```
setTLS(cloud, colNames = NULL)
```

Arguments

cloud	LAS, data.frame, matrix or similar object to be converted or reset.
colNames	optional - character vector. Only used for table-like files. It states the column names - if not set, only the 3 first columns will be converted to XYZ.

Value

LAS object.

Examples

```
cld = matrix(runif(300, 0, 10), ncol=3)
cld = setTLS(cld)
summary(cld)
```

sgmt.ransac.circle *Stem segmentation algorithm: RANSAC circle fit*

Description

This function is meant to be used inside stemSegmentation. It applies a least squares circle fit algorithm in a RANSAC fashion over stem segments. Mode details are given in the sections below.

Usage

```
sgmt.ransac.circle(tol = 0.025, n = 10, conf = 0.99, inliers = 0.8)
```

Arguments

tol	numeric - tolerance offset between absolute radii estimates and hough transform estimates.
n	integer - number of points selected on every RANSAC iteration.
conf	numeric - confidence level.
inliers	numeric - expected proportion of inliers among stem segments' point cloud chunks.

Output Fields

- `TreeID`: unique tree IDs - available only for multiple stems
- `Segment`: stem segment number (from bottom to top)
- `X`, `Y`: circle center coordinates
- `Radius`: estimated circles radii
- `Error`: least squares circle fit error
- `AvgHeight`: average height of stem segments
- `N`: number of points in the stem segments

Least Squares Circle Fit

The circle fit method applied in *TreeLS* estimates the circle parameters from a pre-selected (denoised) set of points by QR decompostion. The optimization criterion for selecting the best circle parameters among several possible candidates is the least squares method, that selects a set of circle parameters that minimize the sum of squared distances between the model circle and its originating points.

RANSAC Algorithm

The **RAN**dom **SA**mple **C**onsensus algorithm is a method that relies on resampling a data set as many times as necessary to find a subset comprised of only inliers - e.g. observations belonging to a desired model. The RANSAC algorithm provides a way of estimating the necessary number of iterations necessary to fit a model using inliers only, at least once, as shown in the equation:

$$k = log(1 - p)/log(1 - w^n)$$

where:

- k: number of iterations
- p: confidence level - desired probability of success
- w: proportion of inliers expected in the *full* dataset
- n: number of observations sampled on every iteration

The models reiterated in *TreeLS* usually relate to circle or cylinder fitting over a set of 3D coordinates, selecting the best possible model through the RANSAC algorithm

For more information, checkout this wikipedia page.

References

Conto, T. ; Olofsson, K. ; Gorgens, E. B. ; Rodriguez, L. C. E. ; Almeida, G. Performance of stem denoising and stem modelling algorithms on single tree point clouds from terrestrial laser scanning. Computers and Electronics in Agriculture, v. 143, p. 165-176, 2017.

Examples

```
### single tree
file = system.file("extdata", "pine.laz", package="TreeLS")
tls = readTLS(file)
tls = stemPoints(tls)
df = stemSegmentation(tls)

head(df)
tlsPlot(tls, df)

### forest plot
file = system.file("extdata", "pine_plot.laz", package="TreeLS")
tls = readTLS(file)

# normalize the point cloud
tls = tlsNormalize(tls)

# map the trees on a resampled point cloud so all trees have approximately the same point density
thin = tlsSample(tls, voxelize(0.02))
map = treeMap(thin, map.hough(min_density = 0.05))

tls = stemPoints(tls, map)
df = stemSegmentation(tls, sgmt.ransac.circle(n=10))

head(df)
tlsPlot(tls, df, map)
```

---

stem.hough *Stem denoising algorithm: Hough Transform*

Description

This function is meant to be used inside `stemPoints`. It applies an adapted version of the Hough Transform for circle search. Mode details are given in the sections below.

Usage

```
stem.hough(hstep = 0.5, max_radius = 0.25, hbase = c(1, 2.5),
  pixel_size = 0.025, min_density = 0.1, min_votes = 3)
```

Arguments

`hstep`	`numeric` - height interval to perform circle search.
`max_radius`	`numeric` - approximately the largest stem cross section radius expected in the point cloud.
`hbase`	`numeric` vector of length 2 - tree base height interval to initiate the circle search.
`pixel_size`	`numeric` - pixel side length to discretize the point cloud layers while performing the Hough Transform circle search.

`min_density` `numeric` - between 0 and 1 - minimum point density within a pixel evaluated on the Hough Transform - i.e. only *dense* point clousters will undergo circle search.

`min_votes` `integer` - Hough Transform parameter - minimum number of circle intersections over a pixel to assign it as a circle center candidate.

`LAS@data` Special Fields

Meaninful fields in the output:

- `TreeID`: unique tree ID of the point - available when a *tree_map* is provided
- `Stem`: `TRUE` for stem points
- `Segment`: stem segment number (from bottom to top)
- `Radius`: approximate radius of the point's stem segment estimated by the Hough Transform - always a multiple of the `pixel_size`
- `Votes`: votes received by the stem segment's center through the Hough Transform

Adapted Hough Transform

The Hough Transform circle search algorithm used in TreeLS applies a constrained circle search on discretized point cloud layers. Tree-wise, the circle search is recursive, in which the search for circle parameters of a stem section is constrained to the *feature space* of the stem section underneath it. Initial estimates of the stem's *feature space* are performed on a *baselise* stem segment - i.e. a low height interval where a tree's bole is expected to be clearly visible in the point cloud. The algorithm is described in detail by Conto et al. (2017).

This adapted version of the algorithm is very robust against outliers, but not against forked or leaning stems.

References

Conto, T. ; Olofsson, K. ; Gorgens, E. B. ; Rodriguez, L. C. E. ; Almeida, G. Performance of stem denoising and stem modelling algorithms on single tree point clouds from terrestrial laser scanning. Computers and Electronics in Agriculture, v. 143, p. 165-176, 2017.

Examples

```
file = system.file("extdata", "spruce.laz", package="TreeLS")
tls = readTLS(file)

### identify stem points
tls = stemPoints(tls, method = stem.hough(max_radius=.2))
plot(tls, color='Stem')
```

stemPoints *Stem points classification*

Description

Classify stem points on a **normalized** point cloud.

Usage

```
stemPoints(las, map = NULL, method = stem.hough())
```

Arguments

las	LAS object.
map	optional - map of tree positions (output from treeMap or treePositions). If omitted, the algorithm assumes las is a single tree.
method	stem denoising algorithm - currently available: stem.hough.

Value

LAS object with *stem_points* signature.

Examples

```
### single tree
file = system.file("extdata", "pine.laz", package="TreeLS")
tls = readTLS(file)
tls = stemPoints(tls)
plot(tls, color='Stem')

### forest plot - check the example given for the stem segmentation function
?stemSegmentation
```

stemSegmentation *Stem segmentation*

Description

Measure stem segments from a classified point cloud.

Usage

```
stemSegmentation(las, method = sgmt.ransac.circle())
```

Arguments

las	LAS object.
method	stem segmentation algorithm - currently available: sgmt.ransac.circle.

Value

data.table of stem segments with *stem_dt* signature.

Examples

```
### single tree
file = system.file("extdata", "pine.laz", package="TreeLS")
tls = readTLS(file)
tls = stemPoints(tls)
df = stemSegmentation(tls)

head(df)
tlsPlot(tls, df)

### forest plot
file = system.file("extdata", "pine_plot.laz", package="TreeLS")
tls = readTLS(file)

# normalize the point cloud
tls = tlsNormalize(tls)

# map the trees on a resampled point cloud so all trees have approximately the same point density
thin = tlsSample(tls, voxelize(0.02))
map = treeMap(thin, map.hough(min_density = 0.05))

tls = stemPoints(tls, map)
df = stemSegmentation(tls, sgmt.ransac.circle(n=10))

head(df)
tlsPlot(tls, df, map)
```

---

tlsAlter *Alter point cloud's coordinates*

Description

Apply transformations to the XYZ axes of a point cloud.

Usage

```
tlsAlter(las, xyz = c("X", "Y", "Z"), bring_to_origin = FALSE,
  rotate = FALSE)
```

Arguments

las	LAS object.
xyz	character vector of length 3 - las' columns to be reassigned as XYZ, respectively. Use minus signs to mirror the axes' coordinates - more details in the sections below.
bring_to_origin	logical - force output to start at c(0,0,0)? If TRUE, removes any geographical information from the output.
rotate	logical - rotate the point cloud to align the ground points horizontally? If TRUE, removes any geographical information from the output. Checkout tlsRotate for more information.

Value

LAS object.

XYZ Manipulation

The xyz argument can take a few different forms, it is useful to shift axes positions in a point cloud or to mirror an axis' coordinates. All axes characters can be entered in lower or uppercase and also be preceeded by a minus sign ('-'), indicating to invert (mirror) the axis' coordinates in the output. Check the *examples* section for a practical overview.

Examples

```
file = system.file("extdata", "pine.laz", package="TreeLS")
tls = readTLS(file)

### swap the Y and Z axes
zy = tlsAlter(tls, c('x', 'z', 'y'))
bbox(zy)
range(zy$Z)
plot(zy, clear_artifacts=FALSE)
rgl::axes3d(col='white')

### return an upside down point cloud
ud = tlsAlter(tls, c('x', 'y', '-z'))
bbox(ud)
range(ud$Z)
plot(zy, clear_artifacts=FALSE)
rgl::axes3d(col='white')

### mirror all axes, then set the point cloud's starting point as the origin
rv = tlsAlter(tls, c('-x', '-y', '-z'), bring_to_origin=TRUE)
bbox(rv)
range(rv$Z)
plot(rv, clear_artifacts=FALSE)
rgl::axes3d(col='white')
```

tlsCrop *Point cloud cropping*

Description

Returns a cropped point cloud of all points inside or outside specified boundaries of circle or square shapes.

Usage

```
tlsCrop(las, x, y, len, circle = TRUE, negative = FALSE)
```

Arguments

las	LAS object.
x, y	numeric - X and Y center coordinates of the area to be cropped.
len	numeric - if circle = TRUE, len is the circle's radius, otherwise it is the side length of a square.
circle	logical - if TRUE (default), crops a circle, otherwise a square.
negative	logical - if TRUE, returns all points outside the specified circle/square boundaries, otherwise returns all points inside the circle/square (default).

Value

LAS object.

Examples

```
file = system.file("extdata", "model_boles.laz", package="TreeLS")
tls = readTLS(file)
plot(tls)

tls = tlsCrop(tls, 2, 3, 1.5, TRUE, TRUE)
plot(tls)

tls = tlsCrop(tls, 15, 10, 3, FALSE, FALSE)
plot(tls)
```

tlsNormalize *Normalize a TLS point cloud*

Description

Fast normalization of TLS point clouds based on a Digital Terrain Model (DTM) of the ground points. If the input's ground points are not classified, the csf algorithm is applied internally.

Usage

```
tlsNormalize(las, res = 0.5, keepGround = TRUE)
```

Arguments

las	LAS object.
res	numeric - resolution of the DTM used for normalization.
keepGround	logical - if TRUE (default), returns a normalized point cloud with classified ground, otherwise removes the ground points.

Value

LAS object.

Examples

```
file = system.file("extdata", "pine_plot.laz", package="TreeLS")
tls = readTLS(file)
plot(tls)
rgl::axes3d(col='white')

### remove topography effect
tls = tlsNormalize(tls, 0.5, FALSE)
plot(tls)
rgl::axes3d(col='white')
```

tlsPlot *Plot TLS outputs*

Description

Plot the LAS outputs of tls functions on the same scene using rgl. Check ?stemSegmentation for usage examples.

Usage

```
tlsPlot(las, sgmt = NULL, map = NULL, treeID = NULL,
  sgmtColor = "yellow")
```

Arguments

las	LAS object - ideally an output from stemPoints.
sgmt	optional data.table - output from stemSegmentation.
map	optional LAS object - output from treeMap.
treeID	optional numeric - single *TreeID* to extract from las.
sgmtColor	optional - color of the plotted stem segment representations.

Examples

```
### single tree
file = system.file("extdata", "spruce.laz", package="TreeLS")
tls = readTLS(file)
tls = stemPoints(tls)
df = stemSegmentation(tls)

tlsPlot(tls, df)

### For further examples check:
?stemSegmentation
```

---

tlsRotate *Rotate point cloud towards a horizontal plane*

Description

Check for ground points and rotates the point cloud to align its ground surface to a horizontal plane (XY). This function is especially useful for point clouds not georreferenced or generated through mobile scanning, which might present a tilted global reference system. Since the coordinates are altered in this procedure, any geographical information is erased from the LAS' header after rotation.

Usage

```
tlsRotate(las)
```

Arguments

las	LAS object.

Value

LAS object.

Examples

```
file = system.file("extdata", "pine_plot.laz", package="TreeLS")
tls = readTLS(file)

### note the tilted ground
plot(tls)
rgl::axes3d(col='white')

### after rotation
tls = tlsRotate(tls)
plot(tls)
rgl::axes3d(col='white')
```

`tlsSample` *Resample a point cloud*

Description

Applies an algorithm that returns a thinned point cloud.

Usage

```
tlsSample(las, method = voxelize())
```

Arguments

`las`	LAS object.
`method`	point sampling algorithm - currently available: `voxelize` or `randomize`

Value

LAS object.

Examples

```
file = system.file("extdata", "pine.laz", package="TreeLS")
tls = readTLS(file)
summary(tls)

## sample points systematically in 3D
vx = tlsSample(tls, voxelize(0.05))
summary(vx)

## sample points randomly
rd = tlsSample(tls, randomize(0.5))
summary(rd)
```

`treeMap` *Map tree occurrences from TLS data*

Description

Estimates tree occurrence regions from a **normalized** point cloud.

Usage

```
treeMap(las, method = map.hough())
```

Arguments

`las`	LAS object.
`method`	tree mapping algorithm - currently available: `map.hough`.

Value

LAS object with *tree_map* signature.

Output

The output is a LAS object with extra fields in the `data` slot. For more details on the output fields checkout `map.hough`'s help page.

Examples

```
file = system.file("extdata", "model_boles.laz", package="TreeLS")
tls = readTLS(file)
plot(tls)

## build a 3D map of tree occurrences
map = treeMap(tls)
plot(map, color='Radii')

## get a 2D representation of the tree map
xymap = treePositions(map)
head(xymap)
```

treePositions *Get unique tree positions from a* tree_map

Description

Extracts the tree XY positions from a *tree_map* LAS object

Usage

```
treePositions(las, plot = T)
```

Arguments

las	LAS object - output from treeMap.
plot	logical - plot the tree map?

Value

data.table of tree IDs and XY coordinates with *tree_map_dt* signature.

Examples

```
file = system.file("extdata", "model_boles.laz", package="TreeLS")
tls = readTLS(file)
plot(tls)

## build a 3D map of tree occurrences
map = treeMap(tls)
plot(map, color='Radii')

## get a 2D representation of the tree map
xymap = treePositions(map)
head(xymap)
```

voxelize *Point sampling algorithm: systematic voxel grid*

Description

This function is meant to be used inside tlsSample. It selects one random point per voxel at a given spatial resolution.

Usage

```
voxelize(spacing = 0.05)
```

Arguments

spacing	numeric - voxel side length.

Examples

```
file = system.file("extdata", "pine.laz", package="TreeLS")
tls = readTLS(file)
print(tls)

### thin point cloud - note the point count
tls = tlsSample(tls, voxelize(0.05))
print(tls)
```

Index