

R workflow for tree segmentation from ALS data

Jean-Matthieu Monnet

2021-10-04

The code below presents a tree segmentation workflow from Airborne Laser Scanning (lidar remote sensing) data. The workflow is based on functions from R packages `lidaRtRee` and `lidR`, and it includes the following steps:

- treetop detection,
- crown segmentation,
- accuracy assessment with field inventory,
- species classification

Steps 1 and 3 are documented in (Monnet et al. 2010; Monnet 2011). The detection performance of this algorithm was evaluated in a benchmark (Eysn et al. 2015).

Licence: GNU GPLv3 / [source page](#)

Many thanks to Pascal Obst  tar for checking code and improvement suggestions.

Material

Field inventory

The field inventory corresponds to a 50 m x 50 m plot located in the Chablais mountain (France) (Monnet 2011, 34). All trees with a diameter at breast height above 7.5 cm are inventoried. The data is available in package `lidaRtRee`.

```
# load dataset from package (default)
data(tree_inventory_chablais3, package = "lidaRtRee")
```

Otherwise you can load your own data provided positions and heights are measured.

```
# import field inventory
fichier <- "chablais3_listeR.csv"
tree.inventory <- read.csv(file = fichier, sep = ";", header = F, stringsAsFactors = TRUE)
names(tree_inventory_chablais3) <- c("x", "y", "d", "h", "n", "s", "e", "t")
# save as rda for later access
# save(tree_inventory_chablais3, file='tree_inventory_chablais3.rda')
```

Attributes are:

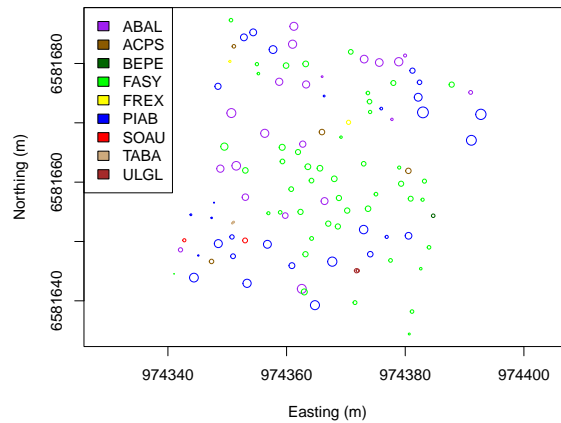
- x easting coordinate in Lambert 93
- y northing coordinate in Lambert 93
- d diameter at breast height (cm)
- h tree height (m)
- n tree number
- s species abbreviated as GESP (GEnus SPecies)
- e appearance (0: missing or lying, 1: normal, 2: broken treetop, 3: dead with branches, 4: snag)
- t tilted (0: no, 1: yes)

```
##           x           y           d           h n           s e t
## 1 974353.3 6581643 37.6 23.6 1 PIAB 1 0
## 2 974351.0 6581648 15.7 13.9 2 PIAB 1 0
## 3 974348.5 6581650 34.2 23.0 3 PIAB 1 0
```

Function `plot_tree_inventory` is designed to plot forest inventory data. Main species are European beech (*Fagus sylvatica*), Norway spruce (*Picea abies*) and silver fir (*Abies alba*).

```
# display inventoried trees
```

```
lidaRtRee::plot_tree_inventory(tree_inventory_chablais3[, c("x", "y")], tree_inventory_chablais3$h,
  species = as.character(tree_inventory_chablais3$s))
```

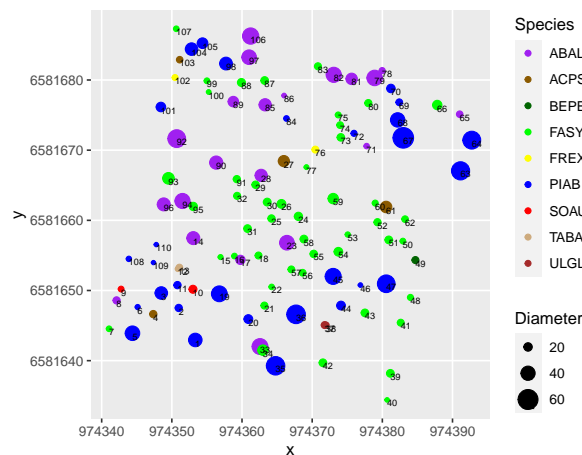


The `ggplot2` package also provides nice outputs.

```
# use table of species of package lidaRtRee to always use the same color for a
# given species
```

```
plot.species <- lidaRtRee::species_color()[levels(tree_inventory_chablais3$s), "col"]
library(ggplot2)
```

```
ggplot(tree_inventory_chablais3, aes(x = x, y = y, group = s)) + geom_point(aes(color = s,
  size = d)) + coord_sf(datum = 2154) + scale_color_manual(values = plot.species) +
  scale_radius(name = "Diameter") + geom_text(aes(label = n, size = 20), hjust = 0,
  vjust = 1) + labs(color = "Species") # titre de la légende
```



We define the region of interest (ROI) to crop ALS data to corresponding extent before further processing. ROI is set on the extent of tree inventory, plus a 10 meter buffer.

```
# buffer to apply around ROI (meters)
```

```
ROI_buffer <- 10
```

```
# ROI limits
```

```
ROI_range <- data.frame(round(apply(tree_inventory_chablais3[, c("x", "y")], 2, range)))
```

Airborne Laser Scanning data

In this tutorial, ALS data available in the `lidaRtRee` package is used.

```
# load data in package lidaRtRee (default)
data(las_chablais3, package = "lidaRtRee")
las_chablais3
```

```
## class      : LAS (v1.2 format 1)
## memory     : 7 Mb
## extent     : 974326, 974408, 6581619, 6581702 (xmin, xmax, ymin, ymax)
## coord. ref. : RGF93 / Lambert-93
## area       : 6803.003 m2
## points     : 92.1 thousand points
## density    : 13.54 points/m2
```

Otherwise, ALS data is loaded with functions of package `lidR`. First a catalog of files containing ALS data is built. Then points located inside our ROI are loaded.

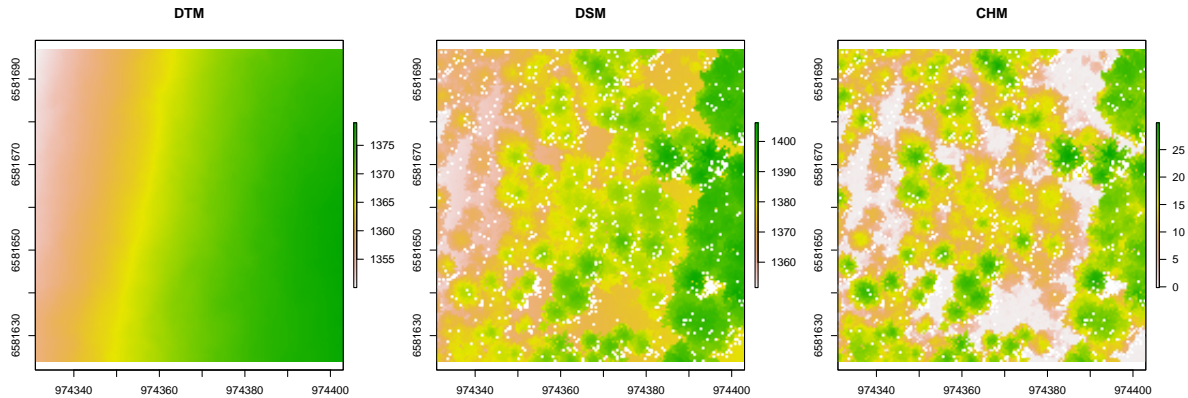
```
# directory for laz files
lazdir <- "../data/tree.detection"
# build catalog of files specifying ALS data
cata <- lidR::readALSLASCatalog(lazdir)
# set coordinate system
lidR::projection(cata) <- 2154
# extract points in ROI plus additional buffer
las_chablais3 <- lidR::clip_rectangle(cata, ROI_range$x[1] - ROI_buffer - 5, ROI_range$y[1] -
  ROI_buffer - 5, ROI_range$x[2] + ROI_buffer + 5, ROI_range$y[2] + ROI_buffer +
  5)
# save as rda for easier access: save(las_chablais3, file='las_chablais3.rda',
# compress = 'bzip2')
```

Data preparation

Digital Elevation Models

From the ALS point cloud, digital elevation models are computed (Monnet 2011, 43–46). The Digital Terrain Model (DTM) represents the ground surface, it is computed by bilinear interpolation of points classified as ground. The Digital Surface Model (DSM) represents the canopy surface, it is computed by retaining in each raster's pixel the value of the highest point included in that pixel. The Canopy Height Model (CHM) is the normalized height of the canopy. It is computed by subtracting the DTM to the DSM.

```
# define extent and resolution of raster
output_raster <- raster::raster(resolution = 0.5, xmn = ROI_range$x[1] - ROI_buffer,
  xmx = ROI_range$x[2] + ROI_buffer, ymn = ROI_range$y[1] - ROI_buffer, ymx = ROI_range$y[2] +
  ROI_buffer)
# terrain model computed from points classified as ground
dtm <- lidR::grid_terrain(las_chablais3, output_raster, lidR::tin())
# dtm_lidaRtRee <- lidaRtRee::points2DTM(lidR::filter_ground(las_chablais3),
# res = 0.5, ROI_range$x[1] - ROI_buffer, ROI_range$x[2] + ROI_buffer,
# ROI_range$y[1] - ROI_buffer, ROI_range$y[2] + ROI_buffer ) surface model
dsm <- lidR::grid_canopy(lidR::clip_roi(las_chablais3, raster::extent(output_raster)),
  output_raster, lidR::p2r())
# dsm_lidaRtRee <- lidaRtRee::points2DSM(las_chablais3, res = 0.5,
# dtm@extent@xmin, dtm@extent@xmax, dtm@extent@ymin, dtm@extent@ymax ) canopy
# height model
chm <- dsm - dtm
```



Visual comparison of field inventory and ALS data

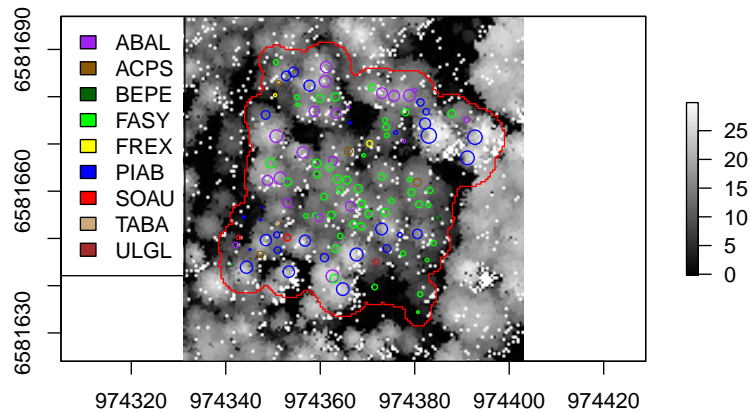
A plot mask is computed from the inventoried positions, using a height-dependent buffer. Indeed, tree tops are not necessarily located vertically above the trunk positions. In order to compare detected tree tops with inventoried trunks, buffers have to be applied to trunk positions to account for the non-verticality of trees.

```
# plot mask computation based on inventoried positions convex hull of plot
mask_chull <- lidaRtRee::raster_chull_mask(tree_inventory_chablais3[, c("x", "y")],
  dsm)
# union of buffers around trees
mask_tree <- lidaRtRee::raster_xy_mask(tree_inventory_chablais3[, c("x", "y")], 2.1 +
  0.14 * tree_inventory_chablais3$h, dsm)
# union of convexHull and tree buffers
mask_plot <- max(mask_chull, mask_tree)
# vectorize plot mask
mask_plot_v <- raster::rasterToPolygons(mask_plot, function(x) (x == 1), dissolve = T)
```

Displaying inventoried trees on the CHM shows a pretty good agreement of crowns visible in the CHM with trunk locations and sizes.

```
# display CHM
raster::plot(chm, col = gray(seq(0, 1, 1/255)), main = "Canopy Height Model and tree positions")
# add inventoried trees
lidaRtRee::plot_tree_inventory(tree_inventory_chablais3[, c("x", "y")], tree_inventory_chablais3$h,
  species = as.character(tree_inventory_chablais3$s), add = TRUE)
# display plot mask
raster::plot(mask_plot_v, border = "red", add = TRUE)
```

Canopy Height Model and tree positions



Tree delineation

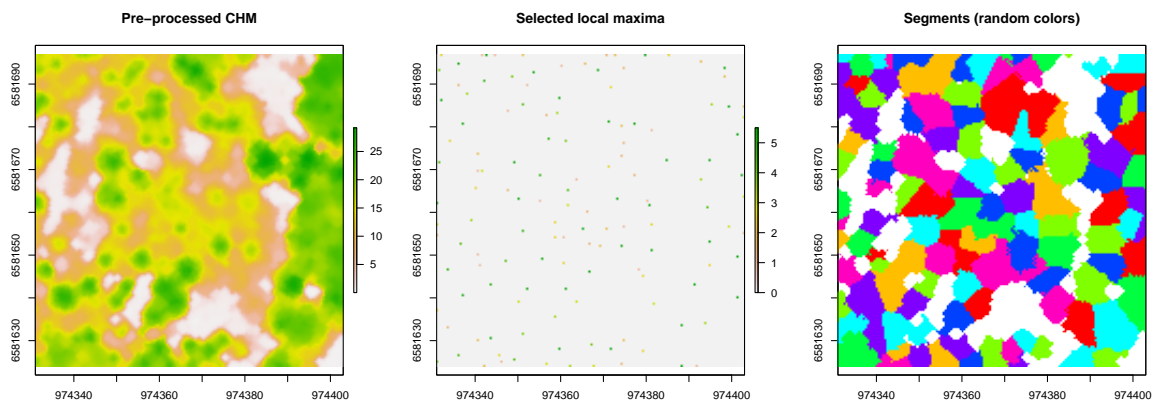
Segmentation

Tree segmentation is performed on the Canopy Height Model by using a general function (`lidaRtRee::tree_segmentation`) which consists in the following steps:

- image pre-processing (non-linear filtering and smoothing for noise removal),
- local maxima filtering and selection for apex (local maximum) detection,
- image segmentation with a watershed algorithm for crown delineation.

The first two steps are documented in Monnet (2011), pp. 47-52.

```
# tree detection (default settings), applied on canopy height model
segms <- lidaRtRee::tree_segmentation(chm)
#
par(mfrow = c(1, 3))
# display pre-processed chm
raster::plot(segms$smoothed_dem, main = "Pre-processed CHM")
# display selected local maxima
raster::plot(segms$local_maxima, main = "Selected local maxima")
# display segments, except ground segment
dummy <- segms$segments_id
dummy[dummy == 0] <- NA
raster::plot(dummy%%8, main = "Segments (random colors)", col = rainbow(8), legend = FALSE)
```



Extraction of apices positions and attributes

A `data.frame` of detected apices located in the plot mask is then extracted with the function `tree_extraction`. Segments can be converted from raster to polygons but the operation is quite slow. Attributes are :

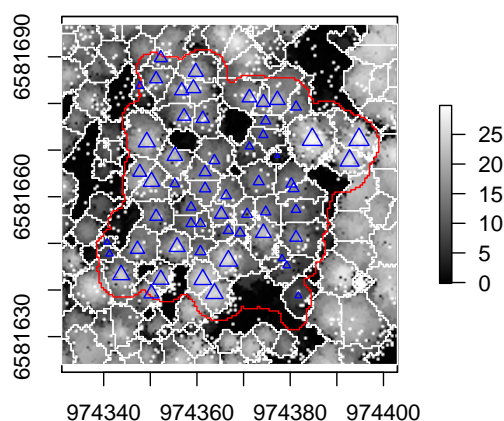
- `id`: apex id
- `x`: easting coordinate of apex
- `y`: northing coordinate of apex
- `h`: height of apex
- `dom_radius`: distance of apex to nearest higher pixel of CHM
- `s`: crown surface
- `v`: crown volume
- `sp`: crown surface inside plot
- `vp`: crown volume inside plot

```
# tree extraction only inside plot mask for subsequent comparison
apices <- lidaRtRee::tree_extraction(segms$filled_dem, segms$local_maxima, segms$segments_id,
  mask_plot)
head(apices, n = 3L)
```

```
##   id    h dom_radius    s      v    sp      vp
## 1  9 21.03      4.5 50.25 724.0775 50.25 724.0775
## 2 10 12.13      2.5 33.50 283.9925 33.50 283.9925
## 3 14 16.15      5.5 50.75 598.5425 50.75 598.5425
```

```
# convert segments from raster to polygons
segments_v <- raster::rasterToPolygons(segms$segments_id, dissolve = T)
# display initial image
raster::plot(chm, col = gray(seq(0, 1, 1/255)), main = "CHM and detected positions")
# display segments border
sp::plot(segments_v, border = "white", add = T)
# display plot mask
sp::plot(mask_plot_v, border = "red", add = T)
# display detected apices
graphics::points(apices$x, apices$y, col = "blue", cex = apices$h/20, pch = 2)
```

CHM and detected positions

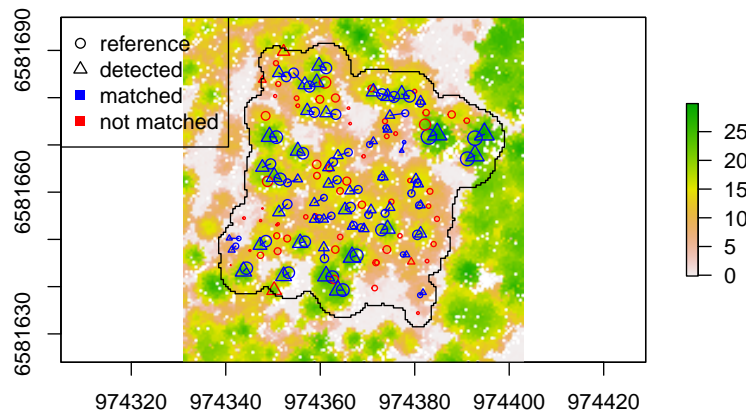


Detection evaluation

Tree matching

To assess detection accuracy, reference (field) trees should be linked to detected apices. Despite the possibility of error, automated matching has the advantage of making the comparison of results from different algorithms and settings reproducible and objective. The algorithm presented below is based on the 3D distance between detected treetops and inventory positions and heights (Monnet et al. 2010). It returns an object with the pairs of matched reference trees and detected apices. The function `lidaRtRee::plot_matched` then plots the results.

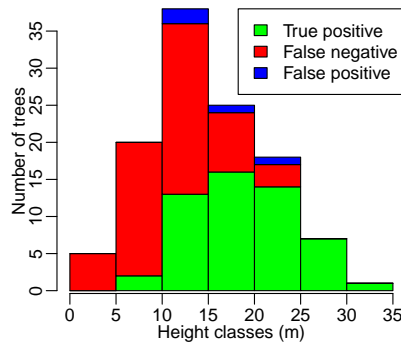
```
# match detected apices with field trees based on relative distance of apices
matched <- lidaRtRee::tree_matching(tree_inventory_chablais3[, c("x", "y", "h")],
  cbind(apices@coords, apices$h))
# display matching results
lidaRtRee::plot_matched(tree_inventory_chablais3[, c("x", "y", "h")], cbind(apices@coords,
  apices$h), matched, chm, mask_plot_v)
```



Detection accuracy

Detection accuracy is evaluated thanks to the number of correct detections (53), false detections (4) and omissions (57). In heterogeneous stands, detection accuracy is height-dependent, it is informative to display those categories on a height histogram.

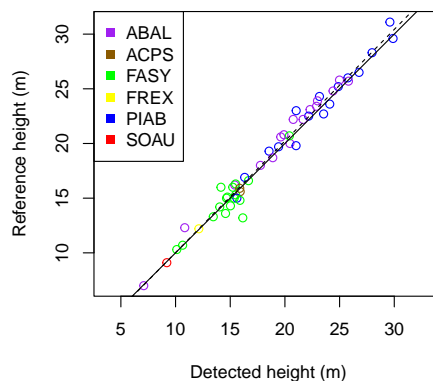
```
# height histogram of detections
detection_stats <- lidaRtRee::hist_detection(tree_inventory_chablais3[, c("x", "y",
  "h")], cbind(apices@coords, apices$h), matched)
```

Height estimation accuracy

For true detections, estimated heights can be compared to field-measured heights. Here, the root mean square error is 1.5m, while the bias is -0.2m. The linear regression is displayed hereafter.

```
# linear regression between reference height and estimated height
height_reg <- lidaRtRee::height_regression(tree_inventory_chablais3[, c("x", "y",
  "h")], cbind(apices@coords, apices$h), matched, species = tree_inventory_chablais3$s)
```



Species classification

The previous steps also output a segmentation of the CHM, i.e. each detected apex is associated to a crown segment in 2D. Next we aim at checking for relationships between the point cloud contained in the segment, and the species of the largest field tree contained in the segment. The processing steps are:

- calculate statistical indices describing the point cloud for each segment,
- extract the highest reference tree from each segment,
- exploratory analysis of the relationship between the indices and the species.

Points in segments

Before computation of point cloud metrics in each segment, the whole point cloud is normalized to convert point altitude to height above ground. Points are then labeled with the id of the segment they belong to. A list of LAS objects corresponding to the segments is then created.

```
# normalize point cloud
lasn <- lidR::normalize_height(las_chablais3, lidR::tin())
# add segment id in LAS object
```



```

lasn <- lidR::add_attribute(lasn, raster::extract(segms$segments_id, lasn@data[,
  1:2]), "seg_id")
# put all seg_id values in ordered list
liste_seg_id <- sort(unique(lasn$seg_id))
# set names of list equal to values
names(liste_seg_id) <- liste_seg_id
# extract point cloud for each segment id in a list
las_l <- lapply(liste_seg_id, function(x) {
  lidR::filter_poi(lasn, seg_id == x)
})

```

Metrics computation

Basic point cloud metrics are computed in each segment (maximum, mean, standard deviation of height, mean and standard deviation of intensity), with the function `lidaRtRee::clouds_metrics` which applies a function to all point clouds in a list, by passing it to `lidR::cloud_metrics`. Please refer to the help of this last function for the expected syntax of the user-defined function to compute metrics.

```

# compute basic las metrics in each segment
metrics <- lidaRtRee::clouds_metrics(las_l, func = ~list(maxZ = max(Z), meanZ = mean(Z),
  sdZ = sd(Z), meanI = mean(Intensity), sdI = sd(Intensity)))
# add segment id attribute
metrics$seg_id <- row.names(metrics)
head(metrics, n = 3L)

```

##	maxZ	meanZ	sdZ	meanI	sdI	seg_id
## 0	8.68	0.6028999	1.299651	84.81373	61.59432	0
## 1	24.21	11.1225458	7.136507	52.19963	45.44458	1
## 2	15.89	8.3338739	4.643465	52.84084	45.61536	2

Merge with reference trees and detected apices

Computed metrics are merged with reference trees and detected apices, thanks to the segment id.

```

# associate each reference tree with the segment that contains its trunk.
tree_inventory_chablais3$seg_id <- raster::extract(segms$segments_id, tree_inventory_chablais3[,
  c("x", "y")])
# create new data.frame by merging metrics and inventoried trees based on
# segment id
tree_metrics <- base::merge(tree_inventory_chablais3, metrics)
# remove non-tree segment
tree_metrics <- tree_metrics[tree_metrics$seg_id != 0, ]
# add metrics to extracted apices data.frame
apices <- base::merge(apices, metrics, by.x = "id", by.y = "seg_id", all.x = T)

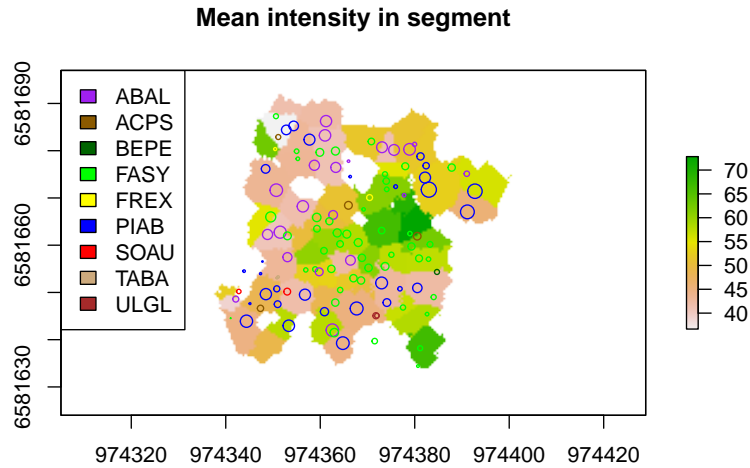
```

Plotting the reference trees with the mean intensity of lidar points in the segments shows that when they are dominant, broadleaf trees tend to have higher mean intensity than coniferous trees.

```

# create raster of segment' mean intensity
r_mean_intensity_seg <- segms$segments_id
# match segment id with id in metrics data.frame
dummy <- match(raster::values(r_mean_intensity_seg), apices$id)
# replace segment id with corresponding mean intensity
raster::values(r_mean_intensity_seg) <- apices$meanI[dummy]
# display tree inventory with mean intensity in segment background
raster::plot(r_mean_intensity_seg, main = "Mean intensity in segment")
# display tree inventory
lidaRtRee::plot_tree_inventory(tree_inventory_chablais3[, c("x", "y")], tree_inventory_chablais3$h,
  species = as.character(tree_inventory_chablais3$s), add = T)

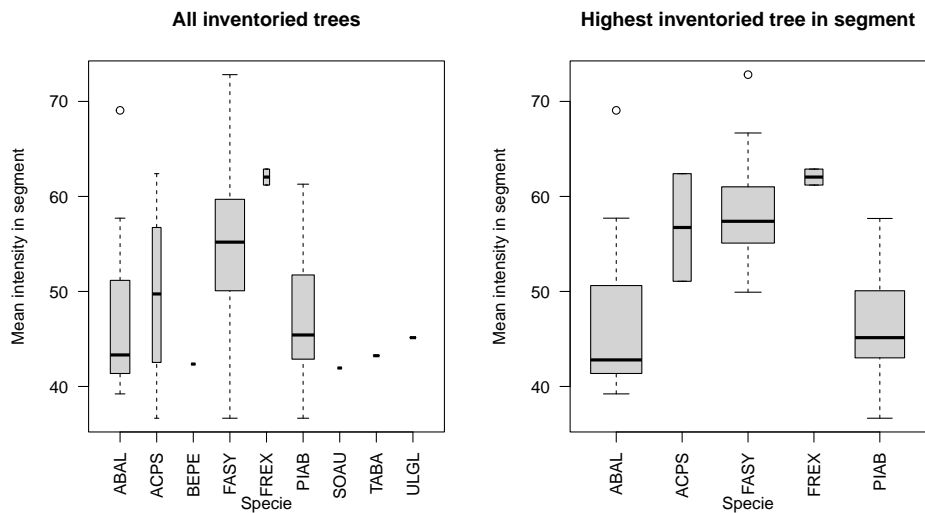
```



Exploratory analysis

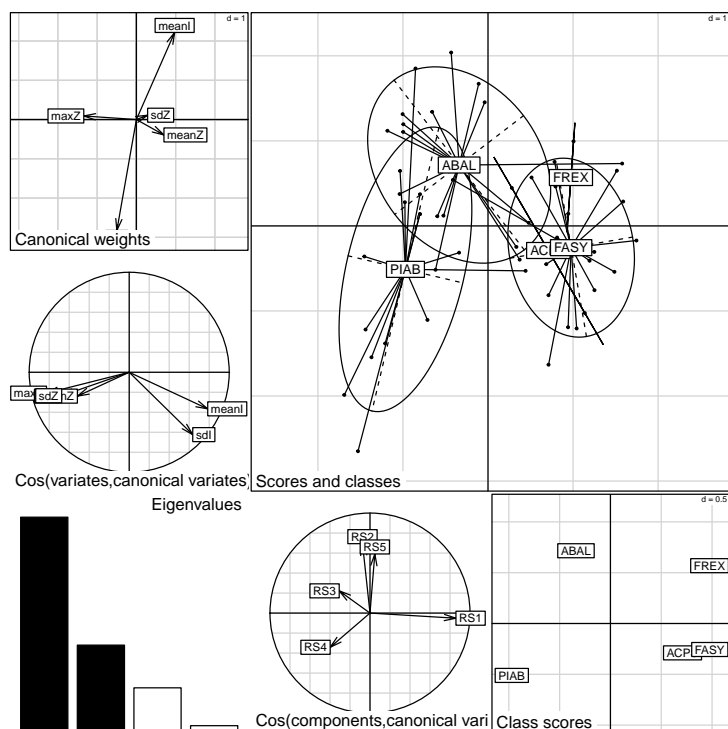
A boxplot of mean intensity in segments per species shows that mean intensity distribution is different between species. The analysis can be run by considering only the highest inventoried trees in each segment, as smaller trees are likely to be suppressed and have smaller contribution to the point cloud.

```
par(mfrow = c(1, 2))
boxplot(meanI ~ s, data = tree_metrics[, c("s", "maxZ", "meanZ", "sdZ", "meanI",
      "sdI")], ylab = "Mean intensity in segment", xlab = "Specie", main = "All inventoried trees",
      las = 2, varwidth = TRUE)
boxplot(meanI ~ s, data = tree_metrics_h, ylab = "Mean intensity in segment", xlab = "Specie",
      main = "Highest inventoried tree in segment", las = 2, varwidth = TRUE)
```



A linear discriminant analysis shows that it might be possible to discriminate between spruce, fir and the deciduous species, thanks to a combination of height and intensity variables.

```
# principal component analysis
pca <- ade4::dudi.pca(tree_metrics_h[, c("maxZ", "meanZ", "sdZ", "meanI", "sdI")],
  scannf = F)
# linear discriminant analysis
lda <- ade4::discrim(pca, tree_metrics_h$s, scannf = F, nf = 2)
plot(lda)
```

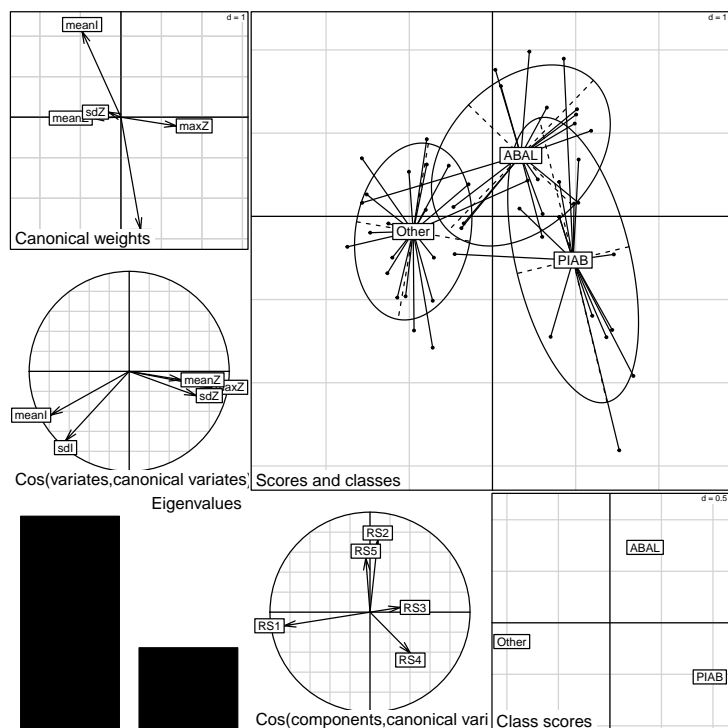


The following line creates a new factor variable **Groups** with three groups: the species PIAB, ABAL and all others together.

```
tree_metrics_h$Groups <- tree_metrics_h$s
levels(tree_metrics_h$Groups)[!is.element(levels(tree_metrics_h$Groups), c("ABAL",
  "PIAB"))] <- "Other"
```

A linear discriminant analysis is performed on this new variable.

```
lda <- ade4::discrim(pca, tree_metrics_h$Groups, scannf = F, nf = 2)
plot(lda)
```



Classification

The function `MASS::lda` is used for prediction purposes. First, the cross-validation option (`CV = TRUE`) is chosen in order to produce a confusion matrix in a prediction case.

```
lda_MASS <- MASS::lda(tree_metrics_h[, c("maxZ", "meanZ", "sdZ", "meanI", "sdI")],
  tree_metrics_h$Groups, CV = TRUE)
# confusion matrix
matrix_confusion <- table(tree_metrics_h$Groups, lda_MASS$class)
matrix_confusion
```

```
##
##          ABAL Other PIAB
## ABAL      9      4      3
## Other     3     18      0
## PIAB      4      1     10
```

```
# percentage of good classification
```

```
round(sum(diag(matrix_confusion))/sum(matrix_confusion) * 100, 1)
```

```
## [1] 71.2
```

```
# confidence interval of the percentage
```

```
binom.test(sum(diag(matrix_confusion)), sum(matrix_confusion))$conf.int
```

```
## [1] 0.5692420 0.8287032
```

```
## attr(,"conf.level")
```

```
## [1] 0.95
```

Then the model is fitted and applied to all segments of the map.

```
# build model
```

```
lda_MASS <- MASS::lda(tree_metrics_h[, c("maxZ", "meanZ", "sdZ", "meanI", "sdI")],
  tree_metrics_h$Groups)
```

```
# apply model to metrics
```

```
metrics$predicted_s <- predict(lda_MASS, metrics[, c("maxZ", "meanZ", "sdZ", "meanI",
  "sdI")])$class
```

```
# apply model to trees
```

```
tree_metrics_h$predicted_s <- predict(lda_MASS, tree_metrics_h[, c("maxZ", "meanZ",
  "sdZ", "meanI", "sdI")])$class
```

To display the classification results, an image of the classified segments is created and the highest field trees in each segment are displayed on top of it. Detentions are correct when:

- purple dots are on purple segments (correct ABAL classification),
- blue dots are on blue segments (correct PIAB classification),
- other colors are on green segments (correct “others” classification).

```
# create image of predicted species
```

```
species <- segms$segments_id
```

```
# replace segment id by predicted species in image
```

```
raster::values(species) <- metrics$predicted_s[match(raster::values(segms$segments_id),
  metrics$seg_id)]
```

```
# remove ground segment
```

```
species[segms$segments_id == 0] <- NA
```

```
# convert to factor raster
```

```
species <- raster::as.factor(species)
```

```
# build raster attribute table rat
```

```
rat <- raster::levels(species)[[1]]
```

```
rat$Species <- levels(metrics$predicted_s)
```

```
levels(species)[[1]] <- rat
```

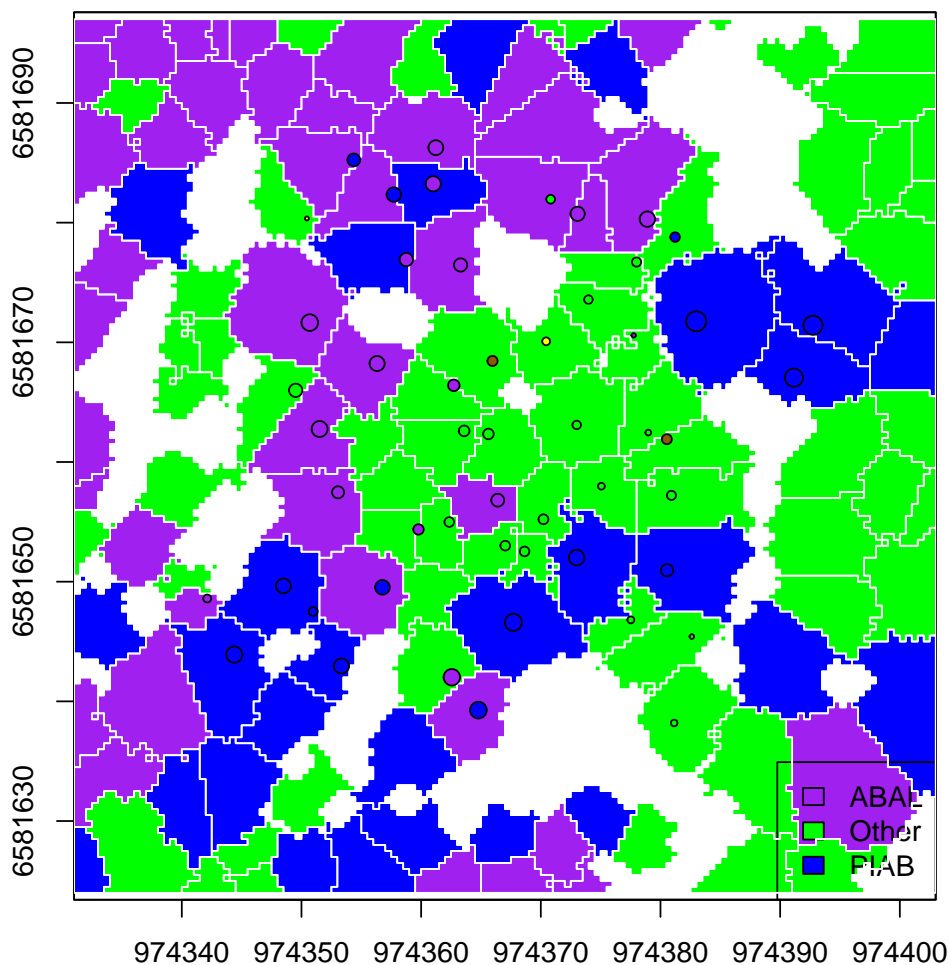
```
# retrieve reference colors
```

```
target_col <- lidaRtRee::species_color()[rat$Species, "col"]
```

```

# set NA color to green
target_col[is.na(target_col)] <- "green"
# display results
raster::plot(species, col = target_col, legend = FALSE)
legend("bottomright", legend = rat$Species, fill = target_col)
sp::plot(segments_v, add = TRUE, border = "white")
lidaRtRee::plot_tree_inventory(tree_metrics_h[, c("x", "y")], tree_metrics_h$h, bg = lidaRtRee::spec
    "col"], col = "black", pch = 21, add = TRUE)

```



Display point cloud

The point cloud can be displayed colored by segment, with poles at the location of inventoried trees.

```

rgl::par3d(mouseMode = "trackball") # parameters for interaction with mouse
# select segment points and offset them to avoid truncated coordinates in 3d
# plot
points.seg <- lasn@data[which(lasn@data$seg_id != 0), c("X", "Y", "Z", "seg_id")]
points.seg$X <- points.seg$X - 974300
points.seg$Y <- points.seg$Y - 6581600

```

```

# plot point cloud
rgl::plot3d(points.seg[, c("X", "Y", "Z")], col = points.seg$seg_id%%10 + 1, aspect = FALSE)
# add inventoried trees
tree_inventory_chablais3$z <- 0
for (i in 1:nrow(tree_inventory_chablais3)) {
  rgl::lines3d(rbind(tree_inventory_chablais3$x[i] - 974300, tree_inventory_chablais3$x[i] -
    974300), rbind(tree_inventory_chablais3$y[i] - 6581600, tree_inventory_chablais3$y[i] -
    6581600), rbind(tree_inventory_chablais3$z[i], tree_inventory_chablais3$z[i] +
    tree_inventory_chablais3$h[i]))
}

```

Batch processing

The following code exemplifies how to process numerous LAS files and extract apices for the whole area with parallel processing. The processing runs faster if data is provided as chunks to the segmentation algorithm, and results are then aggregated, rather than running on the full coverage of the data. In order to avoid border effects, chunks are provided to the algorithm as overlapping tiles. Results are cropped to prevent the same tree from appearing twice in the final results. Tile size and buffer size are important parameters :

- tile size is a trade-off between the number of chunks to process and the amount of RAM required to process a single tile ;
- buffer size is a trade-off between redundant processing of the overlap area, and assuring that a tree in which treetop is located at the border of a tile has its full crown within the buffer size.

Tiles can be processed with parallel computing, within limits of the cluster's RAM and number of cores.

The steps for processing a batch of las/laz files are :

- build catalog of files
- provide tiling parameters
- provide segmentation parameters
- provide output parameters
- set cluster options for parallel processing
- compute the X and Y coordinates of tiles
- parallelize the processing with the package **future**, by sending to the parallel sessions the coordinates of tiles to process, and a function with the instructions to proceed :
 - load tile corresponding to the coordinates
 - compute CHM
 - segment and extract apices
- aggregate list of results

Be aware that in case tree segments are vectorized, some obtained polygons might overlap. The segmentation algorithm might not be deterministic and borders are sometimes not consistent when adjacent polygons are pasted from different tiles.

```

rm(list = ls())
# BUILD CATALOG OF FILES
# folder containing the files
lazdir <- "../data/forest.structure.metrics"
# build catalog
cata <- lidR::readALSLAScatalog(lazdir)
# disable display of catalog processing
lidR::opt_progress(cata) <- FALSE
# set coordinate system
lidR::projection(cata) <- 2154
#
# BATCH PROCESSING PARAMETERS
# tile size to split area into chunks to process
# trade-off between RAM capacity VS total number of tiles to process
tile_size <- 70 # here 70 for example purpose with small area

```

```

# buffer size: around each tile to avoid border effects on segmentation results
# trade-off between making sure a whole tree crown is processed in case its top
# is on the border VS duplicate processing
buffer_size <- 10 # 5 m is minimum, 10 is probably better depending on tree size
#
# TREE SEGMENTATION PARAMETERS
# set raster resolution
resolution <- 1
#
# OUTPUT PARAMETERS
# option to vectorize crowns (set to FALSE if too slow)
out_vectorize_apices <- TRUE
# output canopy height models ? (set to FALSE if too much RAM used)
out_chm <- TRUE
# save chms on disk
out_save_chm <- FALSE
#
# CLUSTER PARAMETERS
library(future)

##
## Attachement du package : 'future'
## L'objet suivant est masqué depuis 'package:raster':
##
##      values
# run on two background sessions
plan(multisession = 2L)

## sequential:
## - args: function (... , envir = parent.frame())
## - tweaked: FALSE
## - call: NULL
#
# COORDINATES OF TILES
# low left corner
x <- seq(
  from = floor(cata@bbox["x", "min"] / tile_size) * tile_size,
  by = tile_size,
  to = ceiling(cata@bbox["x", "max"] / tile_size - 1) * tile_size
) # [1:2]
length_x <- length(x)
y <- seq(
  from = floor(cata@bbox["y", "min"] / tile_size) * tile_size,
  by = tile_size,
  to = ceiling(cata@bbox["y", "max"] / tile_size - 1) * tile_size
) # [1:2]
length_y <- length(y)
# repeat coordinates for tiling while area
x <- as.list(rep(x, length_y))
y <- as.list(rep(y, each = length_x))
#
# PARALLEL PROCESSING
# function takes coordinates of tile as arguments
resultats %<-% mapply(
  # i and j are the coordinated of the low left corner of the tile to process
  FUN = function(i, j) {
    # initialize output

```



```

output <- list()
output$name <- paste0(i, "_", j)
# extract tile plus buffer
las_tile <- lidR::clip_rectangle(
  cata,
  i - buffer_size,
  j - buffer_size,
  i + tile_size + buffer_size,
  j + tile_size + buffer_size
)
# check if points are present
if (nrow(las_tile@data) > 0) {
  # normalization if required
  # las_tile <- lidR::normalize_height(las_tile, lidR::tin())
  # in this example LAS tiles are already normalized
  # compute canopy height model
  chm <- lidaRtRee::points2DSM(las_tile)
  # check all chm is not NA
  if (!all(is.na(raster::values(chm)))) {
    # output chm if asked for
    if (out_chm) {
      output$chm <- raster::crop(chm, raster::extent(
        i, i + tile_size,
        j, j + tile_size
      ))
    }
    # save on disk
    if (out_save_chm) {
      raster::writeRaster(raster::crop(
        chm,
        raster::extent(
          i, i + tile_size,
          j, j + tile_size
        )
      ),
        file = paste0("chm_", i, "_", j, ".tif"),
        overwrite = TRUE
      )
    }
    #
    # tree detection
    segms <- lidaRtRee::tree_segmentation(chm)
    # tree extraction
    apices <- lidaRtRee::tree_extraction(
      segms$filled_dem,
      segms$local_maxima,
      segms$segments_id
    )
    # remove apices in buffer area
    apices <- apices[apices$x >= i & apices$x < i + tile_size &
      apices$y >= j & apices$y < j + tile_size, ]
    # add tile id to apices to avoid duplicates in final file
    apices$tile <- paste0(i, "_", j)
    # convert to vectors if option is TRUE
    if (out_vectorize_apices) {
      # vectorize
      apices_v <- raster::rasterToPolygons(segms$segments_id, dissolve = T)
    }
  }
}

```

```

    # remove polygons which treetop is in buffer
    apices_v <- apices_v[is.element(apices_v$segments_id, apices$id), ]
    # names(apices_v) <- "id"
    # add attributes
    # errors when using sp::merge so using sp::over even if it is probably slower
    # merge(apices_v@data, apices, all.x = TRUE)
    apices_v@data <- cbind(apices_v@data, sp::over(apices_v, apices))
    # save in list
    output$apices_v <- apices_v
  }
  # save apices in list
  output$apices <- apices
} # end of raster is not all NAs check
} # end of nrow LAS check
output
}, x, y, SIMPLIFY = FALSE
) # function applied to the lists of coordinates (x and y)
#
# RESULTS AGGREGATION
# extract results from nested list into separate lists and then bind data
id <- unlist(lapply(resultats, function(x) x[["name"]]))
#
# apices
apices <- lapply(resultats, function(x) x[["apices"]])
# remove NULL elements
apices[sapply(apices, is.null)] <- NULL
# bind remaining elements
apices <- do.call(rbind, apices)
#
# chm
if (out_chm) {
  chm <- lapply(resultats, function(x) x[["chm"]])
  # merge chm
  # no names in list otherwise do.call returns an error
  chm_all <- do.call(raster::merge, chm)
  names(chm) <- id
}
# apices_v
if (out_vectorize_apices) {
  apices_v <- lapply(resultats, function(x) x[["apices_v"]])
  # remove NULL elements
  apices_v[sapply(apices_v, is.null)] <- NULL
  apices_v <- do.call(rbind, apices_v)
  # 1-pixel overlapping in apices_v might be present because image segmentation
  # is not fully identical in overlap areas of adjacent tiles.
}

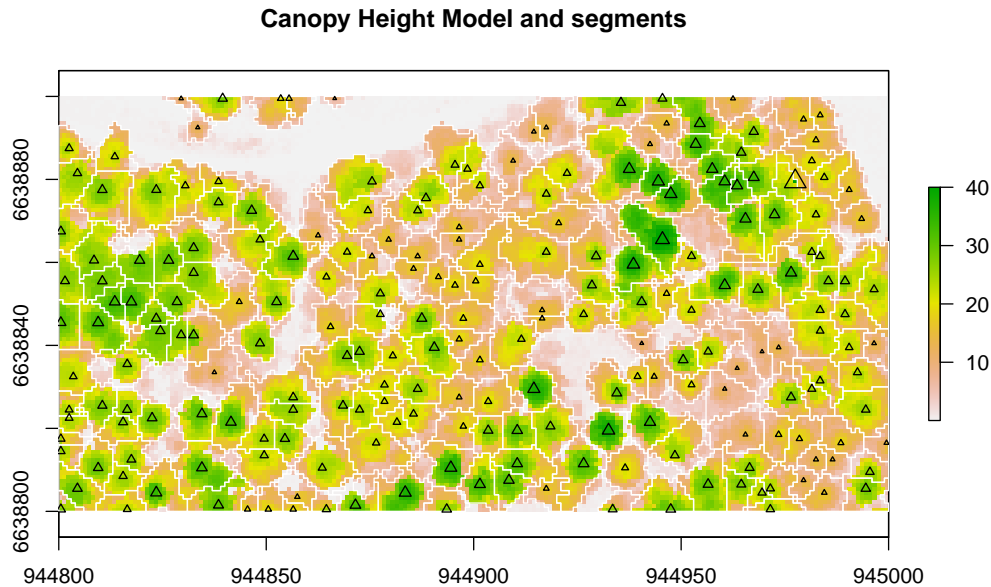
```

The following image displays the results for the whole area.

```

# threshold outsiders in chm
chm_all[chm_all > 40] <- 40
chm_all[chm_all < 0] <- 0
# display chm
raster::plot(chm_all, main = "Canopy Height Model and segments")
# display segments border
sp::plot(apices_v, border = "white", add = T)
# add apices
sp::plot(apices, cex = apices$h/40, add = TRUE, pch = 2)

```



The following lines save outputs to files.

```
# merged chm
raster::writeRaster(chm_all, file = "chm.tif", overwrite = TRUE)
# apices
sf::st_write(sf::st_as_sf(apices), "apices_points.gpkg", "apices_points", delete_dsn = T)
# vectorized apices
if (out_vectorize_apices) sf::st_write(sf::st_as_sf(apices_v), "v_apices_points.gpkg",
    "v_apices_points", delete_dsn = T)
```

References

- Eysn, Lothar, Markus Hollaus, Eva Lindberg, Frédéric Berger, Jean-Matthieu Monnet, Michele Dalponte, Milan Kobal, et al. 2015. "A Benchmark of Lidar-Based Single Tree Detection Methods Using Heterogeneous Forest Data from the Alpine Space." *Forests* 6 (12): 1721–47. <https://doi.org/10.3390/f6051721>.
- Monnet, Jean-Matthieu. 2011. "Using Airborne Laser Scanning for Mountain Forests Mapping: Support Vector Regression for Stand Parameters Estimation and Unsupervised Training for Treetop Detection." PhD thesis, Université de Grenoble. <http://tel.archives-ouvertes.fr/tel-00652698/fr/>.
- Monnet, Jean-Matthieu, Eric Mermin, Jocelyn Chanussot, and Frédéric Berger. 2010. "Tree top detection using local maxima filtering: a parameter sensitivity analysis." In *10th International Conference on LiDAR Applications for Assessing Forest Ecosystems (Silvilaser 2010)*, 9 p. Freiburg, Germany. <https://hal.archives-ouvertes.fr/hal-00523245>.