CSFTools for Airborne LiDAR Data Preprocessing

Jianbo Qi

1. Introduction

Airborne LiDAR datasets have been widely used in forest parameter retrieval. Usually, some common preprocessing operations are needed before using them. These operations include ground filtering, DEM (digital elevation model) generation, DSM (digital surface model) generation and CHM (canopy height model) generation etc. Currently, a lot of commercial or noncommercial tools have been developed. But most of them are separate standalone programs, which cannot be combined together easily to form a complete toolchain, and also are difficult to embed them into a larger project (e.g. your own software).

This user manual aims to introduce a new python-based tool package, which consists of ground filtering, point cloud normalization, DEM/DSM and CHM generation. In addition, an LAI (leaf area index) inversion method (using LiDAR data) is included. This python-based tool makes it possible to embed it into other projects. Since ground filtering usually is a fundamental step for LiDAR point cloud processing and the whole toolchain is based on a ground filtering algorithm named CSF (cloth simulation filter, https://github.com/jianboqi/CSF) (Zhang et al., 2016), which simulates a virtual cloth on the ground inverted, therefore we name this toolchain as **CSFtools**.

2. Installation

This preprocessing tool requires a few python libraries, to make it easier to install, we recommend to use anaconda (https://www.continuum.io/DOWNLOADS, python 3.6+), which has already been integrated with a few scientific computing libraries.
Other libs:

laspy: supporting reading and writing of las file. https://github.com/laspy/laspy

run**:**

pip install laspy

or download the source and run:

python setup.py build

python setup.py install

GDAL: reading and writing raster images.

conda install gdal

joblib: supporting parallel computing for python

pip install joblib

mahotas: computer vision library, supporting watershed transform, etc.

conda config --add channels conda-forge

conda install mahotas

CSF: ground filtering library, more details, see below.

Go to: https://github.com/jianboqi/CSF, and download all the source code:

Under the folder python, run:

python setup.py build

python setup.py install

3. Usage

CSFtools includes several individual tools (commands): csfground for ground filtering, csfdem for DEM/DSM/CHM generation, csfnormalize for point cloud normalization, csfclassify for scalar field based point cloud classification, csflai for leaf area index (LAI) inversion and csfcrown for tree crown extraction from CHM map. The mainly workflow is shown in Fig.1.

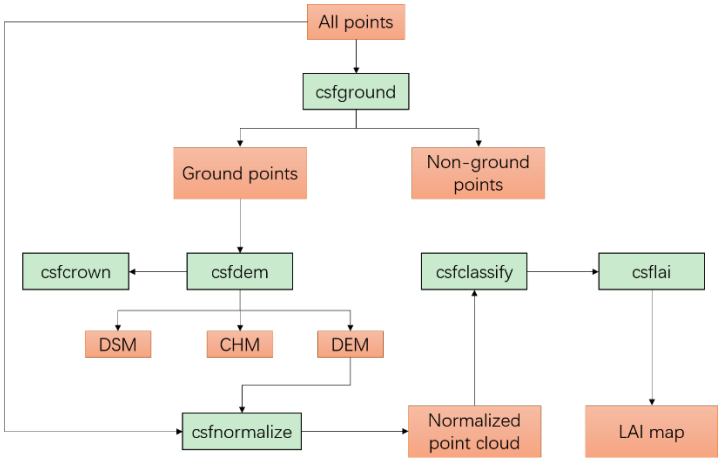

Fig.1 Main workflow of CSFtools

CSFtools uses parallel computing (joblib) heavily to achieve a better performance. Thus, all the above commands (except csfcrown) have a common parameter: **-seg_size**, which is the number of points that are assigned to each core of the computer. The default value is 500000.

3.1 Ground Filtering: csfground.py

To filter a point cloud (las file), run the following command:

python csfground.py -i sample.las -o sample_classified.las

-i: the input las file

-o: the output las file

By default, this command will store the output las file exactly the same as the input las file, only with the scalar field “classification” modified. Ground points are marked with 2, and non-ground points are marked with 1.

There are also parameters to refine the filtering algorithm itself. The default value is applicable to most of forest areas. This method is also adapted to filter LiDAR points of city areas. In that case, it may be necessary to modify the parameter named “**ridigness**”, which stands for how hard the cloth is. All the available parameters are list as follow:

-cloth_resolution: the resolution of the cloth grid. Usually it is more or less the same as average distance between points. Default: 0.5, which is applicable to most situations.
-bSlopeSmooth: In presence of very sharp terrain, set this parameter to “true” to perform a post-processing. Default: true.
-rigidness: the hardness of the cloth. Three options: 1 for rugged terrain, 2 for gentle terrain, 3 for flat terrain. Default: 1.
-classify_threshold: the threshold used to classify the point cloud into ground and non-ground parts. It is based on the distance between each point and the virtual cloth. Default: 0.5. Usually, no need to change.
-save_mode: depending on the selected option (“ground”, “non_ground”, “all”), the output only contains ground points, non-ground points or all points.

3.2 DEM/DSM Generation: csfdem.py

csfdem.py uses a simple gridding and interpolation algorithm to generate a DEM/DSM/CHM.
python csfdem.py -i sample_classified.las –o DEM_file_name -resolution 0.5
-i: the input las file, must have already been classified. It can be the output of the csfground.py.
-o: the output filename (ENVI standard format)
-resolution: the expected resolution of the final DEM/DSM

Optional parameters:
-dsm: the name of the DSM file which is to be created. If this option is provided, it will produce a DSM file along with the DEM.
-chm: the name of the CHM file which is to be created. If this option is provided, it will produce a CHM along with the DEM. Note: -dsm must be provided.
-fillholeofchm: a threshold value used to determine how “holes” in the CHM map may be filled. These “holes” are the low value pixels which are surrounded by higher values within the crown. If the 3 or 4 of height differences between a “hole” and its four neighbor pixels are larger than this threshold, the “hole’ pixel will be filled with average height value of its four neighbor pixels. Default: 3.

-fill_radius: A method to make point cloud denser. It will add new points around each original point with a radius of fill_radius. It has a following parameter fill_num.
-fill_num: How many points will be added for each original point. Default: 3.

3.3 Point Cloud Normalization: csfnormalize.py

The tool normalizes a point cloud using the DEM file. This tool calculates the height difference between point cloud and the DEM, which may be produced by csfdem.py. **Note: when the resolution of the DEM is too coarse, the results may be unreliable.**
python csfnormalize.py –i sample.las –o sample_normalized.las –dem inputdem

3.4 Point Cloud Classification: csfclassify.py

This tool uses a scalar field to classify the point cloud into 2 classes. The results will be stored in scalar field “classification”.
python csfclassify.py -i sample.las -o sample_class.las -field z -value 1.5
-i: input las file.
-o: output las file.
-field: it is used to separate the point cloud. It can be x, y, z, classification, intensity, etc.
-value: the threshold value.

3.5. LAI Inversion: csflai.py

This tool computes the forest leaf area index (LAI) from airborne discrete-return LiDAR data (She-Zhou et al., 2013).
To estimate LAI from LiDAR data, the laser penetration index usually should be calculated according to equation (1)

$$LPI = \frac{N_g}{N_g + N_v} \tag{1}$$

Where LPI is the laser penetration index, N_g is the echo count of echo intensity of ground, and N_v is the echo count of intensity of vegetation. According to beer law,

$$LPI = e^{-G \cdot LAI / \cos\theta} \tag{2}$$

where G is the leaf projection coefficient. θ is the laser beam incident zenith angle. If the leaf angle distribution of canopy is assumed to be spherical distribution, G equals to 0.5. And if the incident angles are ignored (θ =0), then LAI can be derived by $LAI = -\ln(LPI)/G$.

There are mainly three methods to calculate LPI : using echo count (EC), using uncorrected echo intensity (UEI) and using corrected echo intensity (CEI). For EC, N_g is the number of ground echoes, and N_v is the number of vegetation echoes. For UEI, N_g is the total intensity of ground echoes, and N_v is the total intensity of vegetation echoes. For CEI, the intensity of each echo should be normalized before applying it to equation (1). The

normalization is based on equation (3) (García et al., 2010)

$$I_n = I_0 \frac{R^2}{R_S^2} \tag{3}$$

where I_0 is the original echo intensity, and I_n is the normalized echo intensity. R is the range (sensor-target distance). R_S is a reference range, which is usually set as the average flight altitude. Considering the difference of reflectance between vegetation and ground, the LPI is calculated as

$$LPI = \frac{I_g}{I_g + 0.5I_v} \tag{4}$$

where I_g is the normalized intensity of ground, and I_v is the normalized intensity of vegetation.

python csflai.py -i sample_class.las –o sample_output_LAI -resolution 10 -method EC

-i: input las file, it is the result of csfclassify.py, it should already have been classified with Z into lower and higher parts. Lower means ground points, higher means vegetation points. Usually, the classification threshold (-value) is 2. The input of csfclassify.py should be normalized.

-o: the output LAI file. (ENVI standard).

-resolution: the expected resolution of the LAI map. Default: 10

-method: UEI, CEI, EC. Different methods to calculate the laser penetration index of canopy used to estimate LAI. UEI: uncorrected echo intensity. CEI: correct echo intensity. EC: echo count. More details in the paper. Default: EC.

-avgFlightHeight: if CEI is used, this value must be provided. It is the average flight height (altitude).

-originlas: if CEI is used, this value must be provided. It is the original las file (which is not normalized, because CEI need the altitude of each point).

3.6 Tree crown segmentation: csfcrown.py

This tool uses a watershed-based method to extract tree crowns from CHM map. The crown diameter is an average dimeter derived from the crown area.

$$D = 2\sqrt{\frac{A}{\pi}} \tag{5}$$

A is the extracted crown area from CHM map.

python csfcrown.py -i input_chm -o crown.txt

-i: input CHM file

-o: a text file to store the extracted tree crowns (diameter) and tree positions

-height_threshold: a threshold to remove the bushes or grasses. Only pixels with their values larger than this threshold will be considered as tree crown. Default: 2.

-subregion: when the CHM map is too large (e.g. 2km with 0.5 m resolution), it will be divided into several sub regions, and extract crown separately. Default: 1000 (which means

1000 * 1000 pixels).

-window_size: a parameter to find local maxima for watershed algorithm. Usually, it is related to the average size of the crowns. Default: 7, which means 3.5 m for a CHM with 0.5 m resolution.

-color_img: if it is set to “true”, a color image which indicates each extracted crown will be stored for each sub region. Default: false. An example is shown in Fig.2.

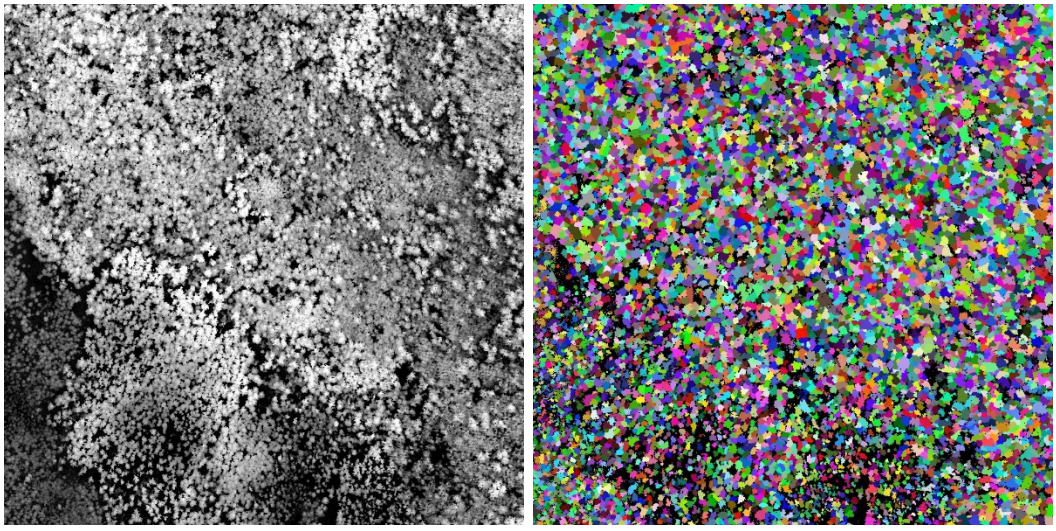

Fig.2 CHM (left), segmented crowns (right)

3.7 Las merge using laspy

Merge several las files into single one, with points from different files can be distinguished by Point Source ID, which starts from 0.

python laspyMerge.py -i *.las -o output.las

Reference:

García, M., Riaño, D., Chuvieco, E., Danson, F.M., 2010. Estimating biomass carbon stocks for a Mediterranean forest in central Spain using LiDAR height and intensity data. Remote Sens. Environ. 114, 816–830. doi:10.1016/j.rse.2009.11.021

She-Zhou, L.U.O., Cheng, W., Gui-Bin, Z., Xiao-Huan, X.I., Gui-Cai, L.I., 2013. Forest Leaf Area Index (LAI) Estimation Using Airborne Discrete-Return Lidar Data. Chin. J. Geophys. 56, 233–242.

Zhang, W., Qi, J., Wan, P., Wang, H., Xie, D., Wang, X., Yan, G., 2016. An Easy-to-Use Airborne LiDAR Data Filtering Method Based on Cloth Simulation. Remote Sens. 8, 501. doi:10.3390/rs8060501