AUTOMATED BUILDING DETECTION USING RANSAC FROM CLASSIFIED LIDAR POINT CLOUD DATA

D.L. Bool, L.C. Mabaquiao, M.E. Tupas, J.L. Fabila

Department of Geodetic Engineering, University of the Philippines – Diliman, Philippines – {deane.bool, luis.mabaquiao.12, marxtupas, johnlouie.fabila}@gmail.com

KEY WORDS: LiDAR, RANSAC, Building Detection, Plane Detection, Python, LAS, Point Cloud

ABSTRACT:

For the past 10 years, the Philippines has seen and experienced the growing force of different natural disasters and because of this the Philippine governement started an initiative to use LiDAR technology in the forefront of disaster management to mitigate the effects of these natural phenomenons. The study aims to help the initiative by determining the shape, number and distribution and location of buildings within a given vicinity. The study implements a Python script to automate the detection of the different buildings within a given area using a RANSAC Algorithm to process the Classified LiDAR Dataset. Pre-processing is done by clipping the LiDAR data into a sample area. The program starts by using the a Python module to read .LAS files then implements the RANSAC algorithm to detect roof planes from a given set of parameters. The detected planes are intersected and combined by the program to define the roof of a building. Points lying on the detected building are removed from the initial list and the program runs again. A sample area in Pulilan, Bulacan was used. A total of 8 out of 9 buildings in the test area were detected by the program and the difference in area between the generated shapefile and the digitized shapefile were compared.

1. INTRODUCTION

In recent years there has been an increasing demand for detailed 3D building descriptions from Airborne Laser Scanning (ALS) also referred to as airborne LiDAR data. Adding the third dimension to planimetric ground plans allow analyzing building heights and their variation, roof shapes and orientation, and visibility studies, etc. (Jochem, et. al, 2009). The 3D description contains the two dimensional planar coordinates along with the addition of a third dimension that represents the elevation of the point.

The emergence of LiDAR technology has paved the ways to different opportunities that has benefited the improvement of the lives of people. It is used in Urban planning, Environmental monitoring, Telecommunications and in Disaster mitigation. The information we get from this technology has impacted our means and considerations for decision making. The problem with this technology is the sheer amount of information, the size of the data. More importantly, the time it takes to process the data into useful information.

LiDAR data is used to classify and separate different buildings from each other. The problem lies in the subjectivity of the classification of these buildings. One building may be composed of various smaller buildings that cannot be immediately distinguished. This may cause problems ahead as it does not take into account the actual distribution of buildings within the composite object.

In this study, a program is created to detect roof planes from LiDAR point cloud data and determine if these planes are part of a single building. The algorithm of RANSAC is used as the basis of obtaining the planar features used to detect the buildings. This study is done to accompany the efforts of the government agencies in simulating the different disasters, primarily flooding, and determining where and how many buildings will be affected in a disaster prone area once the disaster hits.

2. LITERATURE REVIEW

Light Detection and Ranging (LiDAR) technology is applied in airborne laser scanners for an efficient gathering of spatial information. LiDAR data covers information of the ground terrain as wells as man-made structures and vegetation (Liu, 2008). These systems use laser light pulses to measure distance. While the acquisition and use of LiDAR data is effective and significantly less time consuming than conventional data gathering methods, it is important to note that according to Maltezos and Ioannidis (2016), the quality of the plane detection results using LIDAR point clouds is significantly depended by noise, position accuracy, local under-sampling, very large amount of data (having impact on the computational time), low point density, etc. Classified LiDAR data are points already classified to belong to a certain category during the pre-processing stage of the data generation.

There have been numerous procedures and algorithms done to detect a roof plane from a set of 3D points. Some algorithms make use of different input datasets. Some use LiDAR data, Aerial Photographs; LiDAR derived Digital Surface and Terrain Models or a combination of the different sets of data.

Awrangjeb et. al. (2013) presented a rule-based approach for automatic roof extraction. This approach classified the raw LiDAR point cloud into ground and non-ground points. A building mask was generated using the ground points and individual buildings and trees were obtained as clusters of black pixels from the mask. The co-planarity of each non-ground point was tested using the Delaunay neighborhood. After that, planar segments were extracted from the non-ground LiDAR points. To refine the results the authors introduced a rule-based approach. Finally, false planes were removed to get the final set of roof planes. Experimental results showed that the approach missed small buildings and roof planes. Sampath and Shan (2010) presented a solution framework for building roof extraction. It determined the planarity of each LiDAR point based on eigenvalue analysis. Non-planar points were discarded for further processing. After that, it clustered the planar points by using fuzzy k-means approach. The framework achieved good evaluation results. However, the method exhibited high reconstruction error due to removal of LiDAR points near to the plane boundaries. Moreover, the fuzzy k-means clustering algorithm is computationally expensive. Tarsha-Kurdi et al. (2008) applied an extended robust estimation technique on the regenerated LIDAR point cloud using a Hough-Transform and RANSAC algorithm. After converting the original point cloud into a DSM, the missing points were estimated as the mean of the neighboring points. Then a low-pass filter was applied and the raster point cloud was converted to the raw point cloud. As a result, the regenerated points suffer from decreased positional accuracy.

The principle of RANSAC algorithm consists to search the best plane among a 3D point cloud. In the same time, it reduces the number of iterations, even if the number of points is very large. For this purpose, it selects randomly three points and it calculates the parameters of the corresponding plane. Then it detects all points of the original cloud belonging to the calculated plane, according to a given threshold. The input parameters for RANSAC are the 3D point cloud data, tolerance threshold of distance, foreseeable support, and the probability value. 3D point cloud data is a matrix containing columns of X, Y and Z values. Tolerance threshold of distance indicates the highest allowable error for a point to be considered as part of a plane. Foreseeable support indicates the number of points to define a plane. Probability indicates the strength of the iteration process, as it closes to 100%, it is more accurate. Afterwards, it repeats these procedures N times; in each one, it compares the obtained result with the last saved one. If the new result is better, then it replaces the saved result by the new one. (Yang and Forstner, 2016). Determination of basic shapes could be applied on the planes determined by RANSAC. This is done by setting the shapes as proxy of the points it once belong and finding if the planes produces a shape such as cylinders, spheres, etc. This could be done by joining the planes that make up the shapes it defines. (Schnabel, Wahl, & Klein, 2007)

Laspy is a python library for reading, modifying, and creating .LAS LIDAR files. LIDAR data is analogous to RADAR with LASERs, and is short for Light Detection and Ranging. This library provides a python API to read, write, and manipulate one popular format for storing LIDAR data, the .LAS file (Brown & Butler, 2014). It is a computationally efficient workflow developed in Python for processing and analyzing the massive LiDAR point cloud. To exploit the inherent parallelism in analysis of LiDAR point cloud data, a parallel scheme was implemented to allow processing of each "LAS" file in a different process on a multi-core machine. (Kumar, et al., 2016)

This study makes use of the Laspy and LibLas to open, read, write, handle and manipulate LiDAR data within the Python environment.

3. METHODOLOGY

The methodology used is divided into three general workflow. (1).LAS Clipping, (2)Plane Detection and (3)Building Detection. An extension of the study is done by using ArcGIS to visualize and compare the results to pre-existing methodologies (manual digitization).

LAS Clipping
Plane Detection
Building Detection

Figure 1. General Workflow

3.1. LAS Clipping

Classified LiDAR data is used in the study. The data used are classified building points pre-processed using a proprietary software (Bentley Microstation). Cloud Compare software was used to visualize and clip the classified data into smaller sample areas.

3.2. Plane Detection

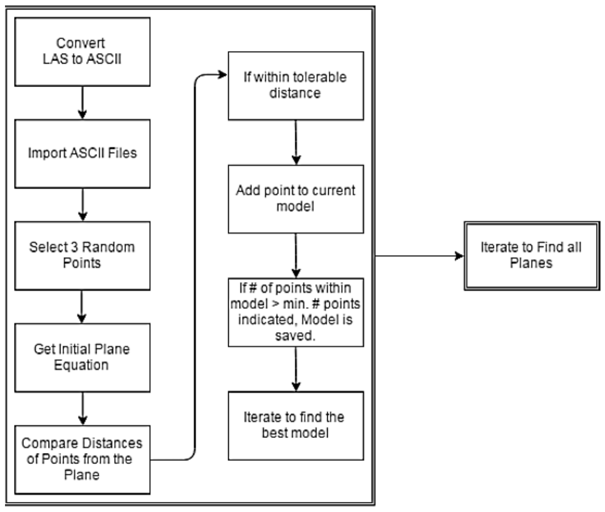
Convert LAS to ASCII
Import ASCII Files
Select 3 Random Points
Get Initial Plane Equation
Compare Distances of Points from the Plane
If within tolerable distance
Add point to current model
If # of points within model > min. # points indicated, Model is saved.
Iterate to find the best model
Iterate to Find all Planes

Figure 2. RANSAC Plane Detection Flowchart

The study created an algorithm for plane detection in Python based on the research or Tarsha-Kurdi. It then extended the research to include the intersection and joining of the results of the plane detection to recreate the roof facet of a building, ultimately detecting the structure as an individual built-up feature. Figure 2 gives the flowchart for the RANSAC plane detection algorithm. The input file needed for the program is a .CSV or an ASCII file. A separate program is used to read, write and convert the .LAS file to a .CSV or an ASCII file using the laspy module of Python. The program starts by importing all the points found within the input file along with the corresponding point numbers. The program selects three (3) random points from the imported set of points. From the selected points, the program creates a plane. Instead of using the Cartesian equation of a plane, the program uses the normal form of the equation of the plane (Tarsha-Kurdi, 2011). The equation is given by the form:

$$\rho = X \cos\theta \cos\varphi + Y \sin\theta \cos\varphi + Z \sin\varphi \qquad (1)$$

Equation 1 presents the normal equation of a plane where θ, φ and ρ are the parameters of the normal passing through the origin. These parameters are constant. This form of the plane equation is used because the distances of the points from the plane obtained are normal to the plane. The distances obtained are the shortest to the plane because it is orthogonal to the plane. After creating the plane from the three randomly selected points, the program computes the distances of every other point from the plane. The distance of a point to the plane is given by:

$$D(X,Y,Z) = X \cos\theta \cos\varphi + Y \sin\theta \cos\varphi + Z \sin\varphi - \rho \qquad (2)$$

Equation 2 gives the function that calculates the distances of a given point from the given plane where X, Y and Z are the three columns of the matrix point list; θ, φ and ρ are the plane parameters. After the distances of every point to the plane are computed, the program checks the number of points within a distance threshold *t*. If the amount of points within the threshold range is within the minimum number of points, *s*, needed to accept the model plane, the plane is saved.

The program iterates for N number of times to determine the best plane. The number of iterations is given by:

$$N = \frac{\log(1-\alpha)}{\log(1-(1-\varepsilon)^s)} \tag{3}$$

Equation 3 describes the computation of the number of iterations needed where N is the total number of iterations, ε is the percentage of observations allowed to be erroneous, α is a minimum probability of finding at least one good set of observations in N trials. It lies usually between 0.90 and 0.99 and s is the minimum number of points necessary to calculate the parameters of the model. For each iteration, the number of points within the range of threshold is determined and compared to the previously saved model. If the new model contains more points, the previous model is discarded and overwritten by the new model. Each set of points in the best model is filtered, saved and written in a separate text file and are deleted from the original set of points. The points lying on the best plane are removed from the initial list of points and saved in a separate text file.

The original RANSAC algorithm determines the best model that describes a given set of points. The result of the algorithm is the parameters of the best model. In the case of the study, the product of the algorithm is the parameters of the best plane that fits the set of points. Points lying on the plane that are significantly distant from the other points may occur so long as the distance of the point to the plane is within the distance threshold. To address this case, the program filters the points before saving and deleting the final list of points from the original set of points. It is also added as another input for the implementation of the program. To filter the set of points, the program selects a point and buffers at a user specified distance; ideally, the maximum distance of each point in a roof plane from one another. Points that fall within the buffered zone of the first point buffers again at the same distance until all possible points are covered in the buffered zone. The program breaks and does not include all other points lying on the same plane but do not fall within the buffer zone from the final list of points. The final list is then removed from the original list and is saved to a separate text file. The program runs again to determine all the possible planes in the input set of points.

Figure 3 presents the pseudocode for the RANSAC plane detection algorithm used in the study.

The program iterates the RANSAC Plane Detection algorithm to find all the best planes within the given data points. Once all the planes have been determined, the program checks the highest point on the initial plane and checks if it is within a user specified distance from an introduced plane. If it does not connect at the first try, it repeats the process but the highest point would come from the introduced plane. If the points fall within the distance, the program will set the initial plane to be the joined point list of the two planes that was processed. The program does not limit the connection between planes to only two planes. It may connect multiple determined planes as long as the specified distance between the highest points of different planes are within the set value.

```
Plane Extraction (1-14) are from Tarsha-Kurdi
1. bestSupport = 0; bestPlane(3,1) = [0, 0, 0]; ~bestPoints(n,4) = [[0,0,0,0],…,[n,n,n,n]]
2. bestStd = 8; i = 0
3. e = 1 - forseeable_support/length(point_list)
4. N = round (log (1 - a)/log (1 - (1 - e) ^3))
5. while (i < N):
6.     j = pick 3 points randomly among (point_list)
7.     pl = pts2plane(j)
8.     dis = dist2plan(pl, point_list)
9.     s = find(abs(dis)<= t)
10.    st = Standard_deviation (s)
11.    if (length(s) > bestSupport or (length(s) = bestSupport and st < bestStd)) then
12.          bestSupport = length (s); ~bestPoints = points_within_t
13.          bestPlan = pl; bestStd = st; endif
14.     i = i+1; endwhile
15. bestPoints(bestSupport,4) = filter(bestPoints,max_dist)
16. exportPoints(bestPoints)
17. point_list = deletePoints(bestPoints from point_list)
18. Restart from 1 while length(point_list) > 0
```

Figure 3. Pseudocode for Plane Detection

3.3 Building Detection

```
Building Detection
0.* N = minimum_pts_for_bldg
1. count = count(planes)
2. files_connected = []
3. i = 0
4. while i < count:
5.   j = 0
6.   main_plane = planes[i]
7.   while j != count and i not in files_connected:
8.     if j not in files_connected:
9.       do_connect = isConnected(main_plane,planes[j])
10.      if do_connect == True:
11.        main_plane = connectPlanes(main_plane,planes[j])
12.        files_connected += j
13.        j = 0
14.      else:
15.        j += 1
16.    else:
17.      j += 1
18.  if len(main_plane) >= N:
19.    exportAsBldg(main_plane)
```

Figure 4. Pseudocode for Building Detection

The rationale behind the idea of connecting the highest points as a factor in determining a building lies within the intuition that roof planes connect at the highest points and not the lowest points. The method was chosen so that the program may still be used on closely structured buildings for it to determine a single building. In addition to the building detection algorithm, the program also checks the number of points belonging to a single building. If the number of points within the detected building is less than a user specified number of points, the program will not consider it as a single building. Figure 4 presents the pseudocode for the feature determination or building detection algorithm.

3.4 Shapefile Generation

The resulting set of detected building points are imported into ArcGIS. The points are saved as shapefiles and the Aggregate Points tool command is used to generate the 2D shapefile that covers all the detected building points. The generated 2D shapefile is compared to the manually digitized polygonal shapefile from the DSM and orthophoto to determine the similarity between the two shapefiles. Similarities between the two polygons are done in terms of area size to compare the two generated shapefiles.

3.5 Data

For the study, the subject area chosen is in Pulilan, Bulacan. The subject area was chosen due to the relatively high density of LiDAR data in the area. The area had an average density of 6 to 8 points per square meter. The LiDAR data was obtained from the UP Phil-LiDAR 1 program. The study area had multiple overlapping flights, which increased the density of the data for the subject area.

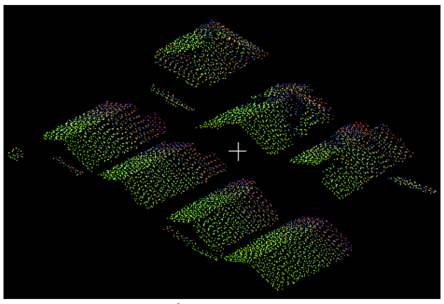
Figure 5. Isometric View of the LiDAR Data in the Test Area of Pulilan, Bulacan

Figure 5 shows the classified LiDAR data of the subject area in front view. The points are classified building points pre-processed using a proprietary software. A sample area is clipped from the LiDAR data to test the program.

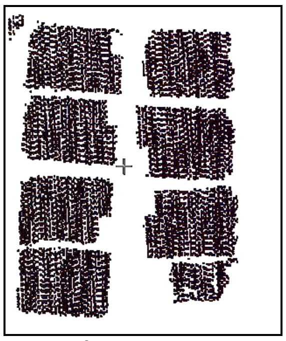
Figure 6. Top View of the LiDAR Data in the Test Area

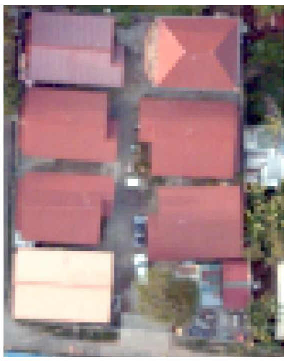
Figure 7. Ortho-photo of Test Area

Location	Pulilan, Bulacan
Total Number of Buildings	9 Buildings
Total Number of Points	9,092 Points
Total Area	1,083.63 sq.m.
LiDAR Density	8.41 Points per sq. m.

Table 1. Test Area Details

4. RESULTS AND ANALYSIS

4.1 Results

LAS Clipping is done before importing the data to the program. The parameters set for the code are 300 points for the minimum acceptance for a model, 0.1m maximum distance of the point to the plane, 0.5m maximum distance for the point filtering and 0.9 for the minimum probability of finding a model. Table 2 shows the summary of parameters used for the sample area. We used the value of 300 points for the minimum acceptance because as per observation and analysis of the sample area, it would be the probable minimum for a recognizable plane based on the density and a plane area. 0.1 and 0.5 meters were used for maximum distances of point to plane and point filtering, respectively, to prevent any excess or insufficiency in the points of the plane, which is the result from the observation of testing the values. We set the probability to the minimum acceptable which is 90% as to lower the number of iterations for a faster processing. The results obtained for the plane detection and building detection for the sample area are shown Figures 8 and 9.

Table 2. Summary of Parameters Used for the Test Area

Parameter	Value
Min. # of points for acceptance	300 Points
Min. distance of point to plane	0.1 meter
Max. distance between points for filtering	0.5 meters
Min. Probability of Finding a Good Set of Observations (α)	0.9 (90%)

Figure 8 shows the planes detected for the sample area. Each color represents the points lying on the detected plane. For the sample area, a total of 33 planes were detected, and from those planes, a total of 8 buildings were identified. 7,896 points were included in the results; these points were determined by the program to belong to a plane. The code ran for a total of 5 hours, 11 minutes, and 8 seconds (05:11:08).

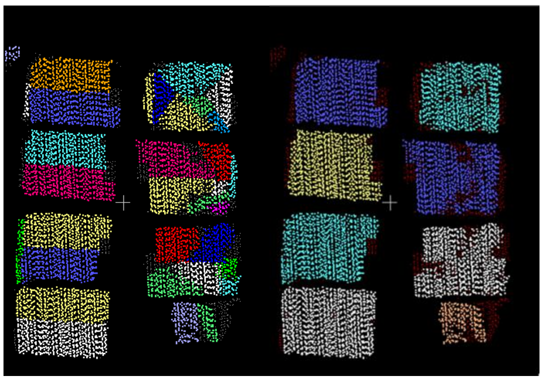
Figure 8. Detected Planes (Left) and Detected Buildings (Right)

Detected Planes	33 Planes
Detected Buildings	8 Buildings
Total Number of Points used	7,869 Points
Total Code Runtime	05:11:31

Table 3. Summary of results on sample area

4.2 Analysis

For the test area, a total of 33 planes were detected. Upon visual inspection of the sample area, 24 planes can be seen. The program resulted to detecting 33 out of 24 planes. This result is attributed to the small planes detected on the upper right building of the test area as seen on Figure 8 (left). Though the detected planes are more than what is expected, the program, thus creating a roof structure, joined these planes together. Eight (8) out of the expected nine (9) buildings were detected. The building on the lower rightmost of Figure 8 (right) was not detected by the program. Only 7,896 points of the 9,092 points were classified as belonging to a specific plane. The remaining points were not classified. The remaining points appear to be relatively distant from one another unabling to program to create a plane that would accommodate the remaining points at the given parameters.

The program ran on a test bench with an Intel Core i5-2410M with 4GB of RAM. The total runtime of the plane detection code is 5:11:08. Figure 10 breaks down the runtime per plane on the sample area.

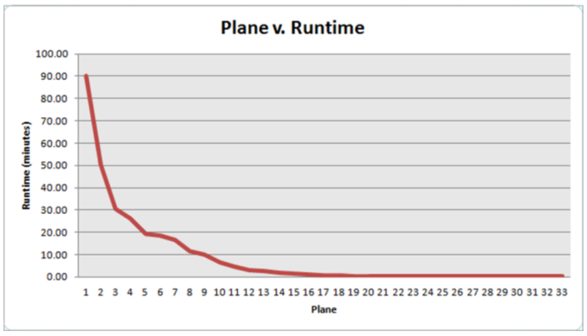

Figure 9. Runtime per plane (in minutes)

The larger planes on Figure 8 (left) were the first to be detected while the smaller planes were detected the last. According to Figure 9, the first plane took 1:30:02 to run while the rest took less than an hour. A pattern appears on the plane detection runtime. The runtime decreases per plane. The succeeding runtime per plane becomes faster due to the decreasing number of points per iteration of a new plane. As for the shapefiles generated and digitized, similarity was tested in terms of their relative area. Table 4 presents the difference in area between the two shapefiles.

Similarity (Area)			
Building	Derived Shapefile (sq.m.)	Digitized Shapefile (sq.m.)	Deviation (%)
Building 1	113.36	148.88	23.86
Building 2	115.07	140.98	18.38
Building 3	120.63	134.67	10.43
Building 4	130.42	160.9	18.94
Building 5	112.03	145.75	23.14
Building 6	127.65	157.86	19.14
Building 7	134.98	146.59	7.92
Building 8	30.96	45.04	31.26

Table 4. Area deviation between shapefiles

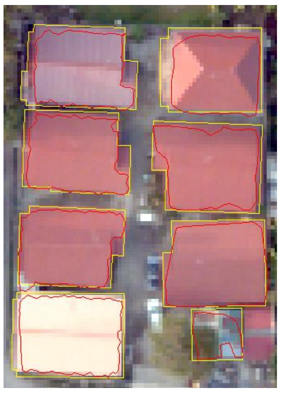

Figure 10. Generated and Digitized Shapefiles from the test area

Table 4 provides the areas of the derived and digitized shapefiles along with its corresponding deviations. The largest deviation can be seen on building 8 with its area being 31.26% different from the digitized shapefile. Building 8 pertains to the lower rightmost building in Figure 8. The difference is attributed to the lack of points detected for that specific building. It is import to note, however, that the digitization of building features is done in the LiDAR-derived Digital Surface Model and not the Ortho-photo. The Ortho-photo only serves as a check for the proper location of the feature on the DSM. The digitized shapefiles in Figure 10 are not necessarily the true shape of the feature on the DSM. For the study, the DSM was not obtained as a primary data for checking; only the Ortho-photo was obtained and used to check and compare the results of the building detection.

Sample Area	Min. points for acceptance	Plane Detection Runtime	Building Detection Runtime	Total Runtime
	300 Points	5:11:08	0:00:23	05:11:31
	1,000 Points	0:25:31	0:12:30	00:37:01

Table 5. Comparison of Runtime at different Min. points

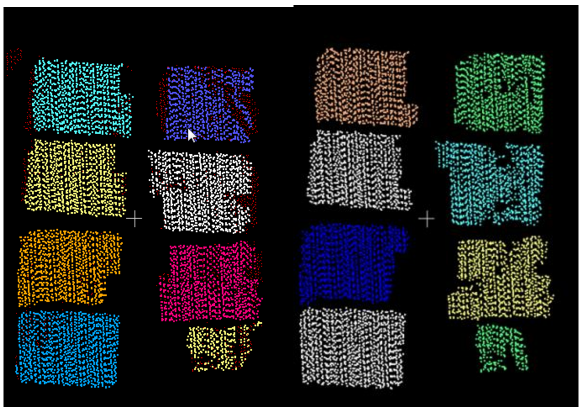

Figure 11. Detected Buildings at different minimum accepted points; 300 points (left) and 1000 points (right)

Using the same test bench at different minimum points for acceptance parameter, the total runtime of the code significantly decreased. The other parameters involved in the code remained the same except for the minimum number of points for acceptance. At 1,000 points the total runtime of the code was only 00:37:01; only 11% of the original runtime. The significant decrease in runtime is attributed to the significant decrease in iterations. Recall that the number of iterations needed for the plane detection is a logarithmic function of the minimum number of points for acceptance. At 1,000 point the number of iterations significantly decreased thus affecting the total runtime of the code. Figures 11 shows the results of the code at different minimum points for acceptance; the same eight buildings were detected and the lower rightmost plane seems to have detected more points for the 1,000 minimum points. Other detected buildings also seem to have more points for the 1,000 minimum points. By visual inspection, the 1,000 minimum points provide a more general result than the 300 minimum points for the sample area.

4.3 Limitations

The result of the plane and building detection is not exactly a plane with defined plane parameters and bounded by lines. It is not geometrically a plane; instead, the result is a list of points that belong to a specific plane. The geometric primitives are still points, not planes. These points take a considerable amount of time to process instead of a simple plane. The result of the Test Bench suggests that the program is heavily dependent on the computing power of the hardware and the parameters set to define the plane. The code is not optimized to run significantly faster on lower platforms. The program is not made to run in parallel with other cores and threads of the CPU. The program has a hard time detecting roof structures with relatively complex plane distribution. Some unconventional types of roof are not detected because the program checks the highest point for each plane. Roof with attics are not immediately detected by the program due to the small sizes of the planes that make up the attic on a roof structure. Lastly, the data utilized in the study has a significantly high density compared to other LiDAR datasets. The parameters are set to accommodate the resolution and density of the data. Other LiDAR dataset with lower density might need to have different corresponding parameters depending on the quality of the data.

5. CONCLUSIONS AND RECOMMENDATIONS

The study is able to develop a methodology that extends the RANSAC algorithm used by Tarsha-Kurdi et. al. to cater to the objective of the study. It was able to develop a RANSAC methodology in Python for plane detection. Depending on the parameters set, the results of the RANSAC algorithm would vary depending on the level of details needed along with the time required to finish the computation. Alongside the RANSAC algorithm, the study develops another methodology for Building Detection that joins the result of the plane detection to create a roof structure. To implement the methodology developed, the study is able to create a program in Python that would ultimately detect buildings from the classified LiDAR data. The derived shapefiles from the detected buildings proved to be significantly similar to the shapefiles digitized from the Ortho-photo and having a maximum deviation of 31% in terms of polygonal area.

The study tried to incorporate the CGAL (Computational Geometry Algorithms Library) project, a pre-built set of algorithms library from C platform bound unto Python to address the geometric computations needed for the program. Unfortunately, the specific algorithm bindings in CGAL, the point/shape detection tool needed for the study was not yet bound to Python.

For this study, a sample area in Pulilan, Bulacan containing a total of 9 buildings with an area of 1080.63 sq.m. is used. The program was able to detect 8 of the 9 buildings and a total of 7,879 points were classified as belonging to a building. It is recommended that for larger and well-spaced building areas like the sample area used, a larger number of minimum points is set due to the smaller number of iterations, short amount of runtime and generalized plane features. For smaller and more detailed areas, a larger density of the area and smaller minimum points for acceptance may be required to get more details of the planes and the buildings.

The authors recommend the extension of the study to use plane as the geometric primitives instead of points so that computations may become less tasking for the computations. Plane outputs (.ply) would also be better since it could be applicable to convert to COLLADA for a better 3D visualization, such as exporting the building outputs into Google Earth Features. Other programming means, or other python modules may do automated generation of shapefiles, as it requires isolation of the boundaries of each building. The point filtering system used is not optimized. It is still not tuned to work for all kinds of dataset. The tuning of the parameters for filtering of points may be improved to further refine the results of the program. Aggregation of points to a polygon shapefile may also be automated by the use of ArcPy or with the aid of the Model Builder, an extension of ArcGIS to further extend the use of the program. In line with the use of CGAL, it is recommended to develop the program in other platforms to test the efficiency and effectivity of the platform, as well as the CGAL library.

ACKNOWLEDGEMENTS

The authors would like to thank the UP PHIL-LiDAR1 Program, especially Engr. Elazegui and Engr. Hingpit for pre-processing and providing the data used for this research.

REFERENCES

Awrangjeb, M., C. Zhang, and C. S. Fraser. 2013. Automatic Extraction of Building Roofs Using LiDAR Data and Multispectral Imagery. ISPRS Journal of Photogrammetry and Remote Sensing, vol. 83, pp. 1 – 18.

Axellson, P. 2000. DEM Generation from Laser Scanner Data Using Adaptive TIN models. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. 33, pp. 110–117.

Brown, G., & Butler, H. 2014. laspy Documentation. Retrieved December 12, 2016, from https://media.readthedocs.org/pdf/laspy/latest/laspy.pdf

Cheng, L., J. Gong, M. Li, and Y. Liu. 2011. 3D Building Model Reconstruction from Multi-view Aerial Imagery and LiDAR Data. Photogrammetric Engineering & Remote Sensing, vol. 77, no. 2, pp. 125–139.

Deschaud, J. E., & Goulette, F. 2010. A Fast and Accurate Plane Detection Algorithm for Large and Noisy Point Clouds Using Filtered Normals and Voxel Growing. France. Retrieved December 15, 2016, from https://hal-mines-paristech.archives-ouvertes.fr/hal-01097361/document

Dorninger, P. and N. Pfeifer. 2008. A Comprehensive Automated 3D Approach for Building Extraction, Reconstruction, and Regularization from Airborne Laser Scanning Point Clouds. Sensors, vol. 8, pp. 7323–7343.

Fischler, M., & Bolles, R. 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. USA. Retrieved December 16, 2016, from http://www.cs.columbia.edu/~belhumeur/courses/compPhoto/ransac.pdf

Forlani, G., Nardinocchi, C. 2001. Building Detection And Roof Extraction In Laser Scanning Data. International Archives of Photogrammetry and Remote Sensing. XXXIV, pp. 319-328.

Gao, J., & Yang, R. 2013. Online Building Segmentation from Ground-based LiDAR Data in Urban Scenes. USA. 2013 International Conference on 3DTV. Retrieved December 16, 2016, from http://www.vis.uky.edu/~jizhou/doc/HouseSeg_LiDAR.pdf

Gubatanga, E., & Ang, M. R. 2014. Automated Filtering Of Non-Terrestrial Points From A LiDAR Point Cloud Data Using a Python Script. Philippines. University of the Philippines - Diliman. Retrieved December 12, 2016.

Hernandez, J., & Marcotegui, B. 2009. Morphological Segmentation of Building Facade Images. 16th IEEE Internation Conference on Image Processing. Retrieved December 16, 2016, from http://cmm.ensmp.fr/~marcoteg/cv/publi_pdf/jorge/FacadeSegmentation/Hernandez_Marcotegui_icip09.pdf

Hu, X., Ye, L. 2013. A Fast and Simple Method of Building Detection from LiDAR Data Based on Scan Line Analysis. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Regina, Canada, Vol. II-3/W1, pp. 7-13.

Joechem, A. et.al. 2009. Automatic Roof Plane Detection and Analysis in Airborne Lidar Point Clouds for Solar Potential Assessment. ISSN 1424-8220. Retrieved from www.mdpi.com/journal/sensors.

Kumar, J., Weiner, J., Hargrove, W., Norman, S., Hoffman, F., & Newcomb, D. 2016. Characterization and Classification of Vegetation Canopy Structure and Distribution within the Great Smoky Mountains National Park using LiDAR. USA. Retrieved December 12, 2016, from http://www.srs.fs.usda.gov/pubs/ja/2016/ja_2016_hargrove_001.pdf

Liu, X. 2008. Airborne LiDAR for DEM Generation: Some Critical Issues. Australia. Monash University, Centre for GIS, School of Geography and Environmental Science, Clayton, Victoria, Australia. Retrieved December 12, 2016, from https://core.ac.uk/download/pdf/11037346.pdf

Lodha, S.K., Kreps, E.J., Helmbold, D.P., Fitzpatrick, D., 2006. Aerial LIDAR data Classification Using Support Vector Machines (SVM). In: Proceedings of the International Symposium on 3D Data Processing, Visualization, and Transmission, Chapel Hill, NC, pp. 567–574.

Maas, H.G.; Vosselman, G. 1999. Two Algorithms For Extracting Building Models From Raw Laser Altimetry Data. ISPRS vol. 54, 153-163.

Matikainen, L. Hyyppa, J. Hyyppa, H. 2003. Automatic Detection Of Buildings From Laser Scanner Data For Map Updating. ISPRS, XXXIV, pp. 218-224.

Perera, S., H. A. Nalani, and Maas, H.G. 2012. An Automated Method For 3D Roof Outline Generation And Regularization In Airborne Laser Scanner Data. ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. I-3, pp. 281–286.

Rottensteiner, F., Trinder, J., Clode, S., Kubik, K. 2005. Automated Delineation Of Roof Planes From LiDAR Data. The Netherlands. International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XXXVI.

Sampath, A. and J. Shan. 2010. Segmentation And Reconstruction Of Polyhedral Building Roofs From Aerial LiDAR Point Clouds. IEEE Trans. Geoscience and Remote Sensing, vol. 48, no. 3-2, pp. 1554–1567.

Schnabel, R., Wahl, R., & Klein, R. 2007. Efficient RANSAC for Point-Cloud Shape Detection. Germany. Blackwel Publishing. Retrieved December 15, 2016, from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.481.1514&rep=rep1&type=pdf

Sohn, G., X. F. Huang, and V. Tao. 2008. Using A Binary Space Partitioning Tree For Reconstructing Polyhedral Building Models From Airborne Lidar Data. Photogrammetric Engineering and Remote Sensing, vol. 74, no. 11, pp. 1425–1440.

Tarsha-Kurdi, F., T. Landes, and P. Grussenmeyer. 2008. Extended RANSAC Algorithm for Automatic Detection of Building Roof Planes from LiDAR Data. The Photogrammetric Journal of Finland, vol. 21, no. 1, pp. 97–109.

Yang, M., & Forstner, W. 2010. Plane Detection in Point Cloud Data. Germany. Retrieved December 15, 2016, from http://www.ipb.uni-bonn.de/pdfs/Yang2010Plane.pdf

Zhou, Q., Neumann, U., 2008. Fast and extensible building modelling from airborne lidar data. Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 7:1–7:8.

Zuliani, M. 2008. RANSAC for Dummies. USA. Retrieved December 15, 2016, from http://old.vision.ece.ucsb.edu/~zuliani/Research/RANSAC/docs/RANSAC4Dummies.pdf

Revised August 2018