Software Testing Plan

April 1, 2022

Team LumberHack

Sponsor:

Dr. Andrew J. Sánchez Meador

Mentor:

Melissa D. Rose

Team Members:

Matthew Flanders

Jenna Pedro

Thomas Whitney

Colin Wood

Version: 1.0

Table of Contents

Introduction

Forest ecosystem health is at the center of many large-scale environmental problems. Efforts and policies that promote awareness and environmental health are crucial to climate change and reducing carbon emissions. These policies are based on scientific research. The sponsor for this project, Dr. Andrew J. Sánchez-Meador, is an Associate Professor at the School of Forestry, as well as the Executive Director at the Ecological Restoration Institute at Northern Arizona University. His research focuses on using quantitative forest ecology for multi-scale forest ecology and restoration projects. Dr. Sánchez-Meador and other forest researchers use tools such as lidar and Mobile Laser Scanning (MLS) to collect informative-data for understanding and interpreting the health of large forested areas. Lidar is a remote sensing method that uses light pulses combined with Global Positioning System (GPS) data to create precise, three-dimensional point clouds replicating the shape and characteristics of trees and their surrounding environment. MLS is a relatively new form of lidar surveying.
Current tools that analyze MLS data are lacking in a few ways such as:

- No focus on mobile lidar
- Steep and complex learning curves
- Programming knowledge is usually required
- Visualization can be improved upon
- No graphical user interfaces (GUI's)

The application tries to address these issues and make working with MLS data more accessible for forest researchers.

The envisioned solution of this project will serve as a resource for researchers and ecologists to analyze their own MLS data. The application will be created in the R programming language with C++ backend support. It will include a user interface that allows users to upload and clean their MLS data, use individual tree segmentation algorithms, classify points, and then summarize and report useful information back to the user.

As the team continues development on the final product, it is crucial that software tests are implemented. Software testing is the process of verifying that the software does what it is intended to do. Testing will also serve as a way to verify that the requirements of the software are being met. The team will use testing as a way to ensure bug-free code, as well as a smooth user experience. Tying testing into the development process will provide a baseline for future development and expansion of the application.

The team has split the testing process into three different software test types: Unit, integration, and usability testing. Unit tests serve as a way to test small, isolated chunks or pieces of code. Unit tests do not rely on eternal systems but rather test one specific function at a time. In this case, unit testing will take a closer look at the individual modules that make up the final MLS data processing application. Integration testing will look at and ensure that the major modules that make up the system are correctly working together. Unit and integration testing work together to test isolated units of code on their own, and then test that individual units integrate together and work correctly with other units where needed. Finally, the team will outline the usability testing process. Usability testing is a hands-on testing approach that will focus on user interaction with the software. Usability testing will make sure that end-users can find and use the intended functionality of the application.

In the context of this project, the team’s following proposed plan accounts for requirements that have been outlined by the sponsor of the project. Unit and integration tests will be conducted to meet and verify both functional and non-functional requirements necessary to provide researchers with accurate MLS data analysis. Usability testing is tailored to end-users in forestry related fields of study and research. It is the goal of this document to outline how testing will be conducted to ensure that the application meets these requirements.

Unit Testing

In order to meet specified requirements, software testing will be used. The first type of testing covered in this document is the unit test. Unit testing looks at individual functions of a piece of software to test the correctness of isolated chunks of code. In the context of this application, the team will be testing how the major modules of the application function and how each unit related to a module will be tested. Each unit will be looked at in detail and the purpose of testing it will be described.

The first module to be tested is file uploading. With this module, users may upload one or many files up to 10 gigabytes (gb) of .las and .laz type. All of the unit tests related to file upload will be conducted using the R library *testthat*. This library is a framework that implements functions for unit testing in R. In this module, files are selected by the user and uploaded to the application. There are two units to be tested in this module. The first unit test will test that files are of the desired .las or .laz file types. This test will verify that no other file types are uploaded and that instead the user receives a reminder message to only upload .las or .laz files. The other unit to test in this module is that the total size of files uploaded does not exceed 10gb. This will test that any file size greater than 10gb will not be uploaded to the application. Edge cases here will be accounted for using the *testthat* library.

The second module to be tested is data cleaning. With this module, users can clean their selected data from the many files that they have uploaded. Once files are uploaded and selected by the user, they can click on the ‘Clean Data’ button. The unit test will test that the files have been cleaned. This test will verify that the file is cleaned and that the user receives a pop up message that the data is cleaning.

After data has been cleaned, the third module to be tested is plotting the data. With this, users will be able to visualize the data file that has been selected and cleaned. Users can click on the ‘Draw Point Cloud’ Button which shows from a RGL viewer for 3D point clouds. There are 5 units to be tested in this module. The first unit test is to test if the ground has been classified with cloth simulation. This test will verify that the point clouds have been turned upside down, then use the cloth simulation filter, and lastly, classify the ground. The second unit to test is if the height has been normalized. This test will verify that the dataset has been normalized with the ground at 0 and that the user receives a pop up message that the height is normalizing. The third unit to test is that the noise has been classified. This test will verify that noise points have been identified so that it can be filtered out when the data is being plotted and that the user receives a message that it is classifying noise. The next unit to test is that the points have been filtered. This will verify that the outliers have been removed and not show on plotted data. The users will also receive a pop up message that it is filtering points. The last unit to be tested in this module is the plot of the point cloud. This will verify that all of the previous steps have been done correctly and therefore create a visual of the point cloud.

The fourth module to be tested is to draw a slice of the tree. With this module, users will be able to get a slice of a tree and see a visual of that slice. Users can click on the 'Draw Slice' button which shows the RGL viewer for slice view. The unit to be tested is that with the file selected from the user, the plot of slice will be created when the button is pressed. This will verify that individual tree locations have been identified, segmented, sliced, and then plotted. Users will also receive a pop up message that the data is slicing and segmenting trees.

The fifth module to be tested is the circle shape fitting with RANSAC. With this module, users will be able to get the best circle fit and calculate useful characteristics about each tree. The unit to be tested is that the csv file gets returned correctly with the best overall count and best circle fit that includes x and y coordinates and radius. This will verify that the number of tree IDs that was sent out gets returned back the same.

The last module to be tested is the data summary table. With this module, users will be able to view characteristics about individual trees. The unit to be tested is that important information about individual trees can be displayed in a 2D table based on RANSAC fitting. This will verify that the users can see the derived data products such as center point, circumference, radius, diameter, and directionality all in one table. Each of these unit tests serves as a way to verify that the individual unit is serving its intended purpose. The next section will discuss how these units and modules tie together in the end application.

Integration Testing

Integration testing serves the purpose of ensuring that the major subcomponents of a software package interact with each other as expected. In this case, there are three major subcomponents. The first is the Shiny front-end, which is responsible for accepting input from the user, displaying output to the user, and driving the other two subcomponents. The second major subcomponent is the lidR package, which contains much of the functionality needed to parse, clean, and normalize the incoming lidar data. The third major subcomponent consists of the C++ library, which is responsible for various computationally intensive processes throughout the workflow. The interfaces between these three subcomponents and the testing designed to ensure their proper interaction will be described next.

The first important interface exists between the Shiny module and the lidR package. After a user uploads their lidar file(s), the file(s) has to be processed using functions from the lidR package. In order for this processing to work correctly, the file that is uploaded must be of the correct format. The software already has one measure in place to help ensure that the file is the correct format, namely a selective upload feature that only allows files with either an .las or .laz file extension to be uploaded. However, a file may have been given an improper extension, so this is not an assurance that the file can be processed by lidR. Furthermore, the official lidar binary format has gone through several revisions and although a file may be a lidar file, it might not be of a version that the lidR package accepts. Thus, the first important test in the workflow is one of assuring that the uploaded file is of a correct format. This will be accomplished by using lidR's "las_check" function, and reading the version number from the lidar file header to ensure that the file version is interoperable with lidR.

Also taking place at the interface between Shiny and the lidR package are the set of function calls used to filter, clean, and normalize the lidar point cloud. All of these operate on a lidar object that the lidR package instantiates when the file is first uploaded and parsed, so it can be non-obvious if an error occurs because there is no output from these functions, only side effects to the object. Although the lidR package does have many helpful messages for various types of errors during these steps, the error messages can't be assumed to cover all problems, and furthermore, these error messages are useless in a production setting because the user can not be expected to debug the program, or even find the error messages wherever they are sent. Thus, it's important to have additional checks after each of these processing stages. This will be accomplished by manually inspecting the las object after each processing step to ensure that a satisfactory amount of points remain after cleaning and filtering, and that ground normalization has not resulted in impossible or unexpected values.

The second important interface exists between the Shiny app and the C++ library routines that are used for shape fitting and other parallel processing tasks. The first step in the back-and-forth between these two submodules is the transfer of data from Shiny to the appropriate C++ functions. The C++ library expects the point data in comma-separated values format, with certain types and a certain number of columns. Because this csv file is generated within the R source code and not by a user, there is little chance that the formatting will be incorrect. However, because even the smallest error in the data could invalidate all results, the csv file needs to be validated before processing begins. This will be implemented by a C++ function that parses the incoming csv file, checking for the proper number of columns, the proper data types, and that no values are missing, before the data is handed off for processing.

A similar situation exists after processing has completed in C++: the data need to be passed back to Shiny in the csv format. Here again, an erroneous csv file could disrupt the correct display and/or interpretation of the data in the front-end. A similar validation function to the one discussed above will be implemented to ensure that the csv data is complete, of the correct dimensions, and of the correct types. Integration testing this application will make sure that each of the modules is correctly communicating and working with the others. Next, the usability testing plan is outlined.

Usability Testing

The purpose of usability testing is to ensure that end users of the product are able to effectively interact with the software and access its features. The goal of usability testing is to improve end user experiences by making changes to the user interface or by adjusting the workflow of the product so that it becomes more intuitive for the end users. The intended end users are forestry professionals, or forestry students, who are working with lidar data and may have experience working with other lidar specific software such as cloud compare or R packages such as lidR. While some of these users may be technologically proficient in working with lidar data the product aims to be straightforward enough so that end users with little experience are able to effectively interact with the product.

To evaluate the usability of the product the team plans to observe users as they interact with the user interface and then ask them to fill out a questionnaire at the end of the testing session. The questionnaire will have a section for each activity that will ask them to them rate their experience on a scale of one to ten as well as have them answer more open-ended questions where they will be prompted to elaborate on what they liked about the activity, what they did not like about the activity, and what they would like to see changed. The team plans to have end users from the forestry department use the product multiple times so that improvement in the end user experience can be tracked. The team plans to have at least two separate groups of end user testers so that first impressions can be observed from one group and work to improve the first impressions for the next group of testers, and also plan to have these users try the product at least two times to see if improvements were made in the product. The following activities will be evaluated with all end users:

- Launch the application
- Load a lidar file
- Clean the lidar point cloud data
- View the point cloud
- Segment trees in the point cloud
- Calculate tree attributes with default RANSAC parameters
- View results of tree attributes

- Adjust RANSAC parameters

Once a testing session has concluded observations and the feedback given on the questionnaire will be used to make improvements on the user experience for the given activity. Usability testing will take about three weeks, and the team hopes to see weekly improvements in user questionnaire ratings and in observed ease of use.

Conclusion

This project has the potential to improve the experiences of many forestry researchers and the current lidar workflows. For mobile lidar, this application will be an all inclusive way to process and generate useful information about trees within mobile lidar scans. With any product, testing ensures the reliability of the product from a technical and end user perspective. In the case of this application, unit testing will verify that individual modules accurately produce expected results. Stress testing each unit of the application makes sure that end results are reliable and accurate for research purposes. Integration testing brings the individual modules together to test that they work correctly with one another. With this application, las/laz data flows through each of the modules from upload until it is fully cleaned and interpreted. Integration testing verifies that each of these steps work together and correctly build off of one another. Finally, the application will be put through usability testing. Usability testing makes sure that the end users of the product can effectively and efficiently use the product for its intended purposes. In this case, forest researchers will be selected to use their own mobile lidar data with this application. Each of these testing methods will serve their own purpose for verifying that all parts of the end application are working as intended. The implementation of requirements is going well and the team is working on testing all of these functions. The team is confident that these tests will be implemented into the final product and that this application will be a useful tool for improving the MLS data workflow.