

PROTOTYPING AUTOMATED FRAMEWORK FOR ASSET EXTRACTION AND CHARACTERIZATION FROM MOBILE LIDAR DATA

Final Report (Appendices)

SPR 866

by

Erzhuo Che, PhD, Assistant Professor (Senior Research)
Jaehoon Jung, PhD, Assistant Professor (Senior Research)
Sahar Kamalou, Graduate Research Assistant,
Kenneth Kenney, Graduate Research Assistant

Oregon State University
101 Kearney Hall, Corvallis OR 97330

Michael J. Olsen
EZDataMD LLC

for

Oregon Department of Transportation
Research Section
555 13th Street NE, Suite 1
Salem OR 97301

and

Federal Highway Administration
1200 New Jersey Avenue SE
Washington, DC 20590

August 2026

ACKNOWLEDGEMENTS

The research team appreciates the efforts and insights of the TAC in providing feedback on the project. Individuals include Jennifer Lanzarotta, Chris Harris, Chris Wright, Mike Woodford, Eric Arndt, Jeff Stewart, Jon Lazarus, Josh Roll, and Rhonda Dodge. The research team also thanks Dr. Jiaxing Zhang and Dr. Dae Kun Kang for their help in testing the software and developing the user manual, as well as Dazzle Fang for some of the software design and development. It is with our condolences that we acknowledge the loss of Lisa Letney from ODOT, who served a brief time on the TAC.

DISCLAIMER

This document is disseminated under the sponsorship of the Oregon Department of Transportation and the United States Department of Transportation in the interest of information exchange. The State of Oregon and the United States Government assume no liability of its contents or use thereof.

The contents of this report reflect the view of the authors who are solely responsible for the facts and accuracy of the material presented. The contents do not necessarily reflect the official views of the Oregon Department of Transportation or the United States Department of Transportation.

The State of Oregon and the United States Government do not endorse products of manufacturers. Trademarks or manufacturers' names appear herein only because they are considered essential to the object of this document.

This report does not constitute a standard, specification, or regulation.

Dr. Che has financial interests in EZDataMD LLC, a company which commercializes point cloud analysis technology related to this research. The conduct, outcomes, or reporting of this research could benefit EZDataMD LLC and could potentially benefit Dr. Che.

TABLE OF CONTENTS

APPENDIX A: QUICK USER GUIDE FOR GROUND FEATURE EXTRACTION SOFTWARE PROTOTYPE.....	1
BACKGROUND.....	2
SOFTWARE SETUP	3
BASIC PROGRAM SETUP	4
COMPONENTS OF A PROJECT.....	7
GRAPHICAL USER INTERFACE (GUI).....	14
CONTRIBUTIONS.....	18
APPENDIX B: QUICK USER GUIDE FOR GIS TOOLBOX PROTOTYPE	19
INSTALLATION	20
<i>ArcGIS Pro Installation.....</i>	<i>20</i>
<i>ArcGIS Pro Python Environment.....</i>	<i>20</i>
<i>Configure Python Libraries.....</i>	<i>20</i>
ADDING AND RUNNING TOOLBOX	23
<i>Horizontal Toolbox</i>	<i>23</i>
<i>Vertical Toolbox.....</i>	<i>25</i>
<i>Cross section Toolbox</i>	<i>28</i>
<i>Visualization Toolbox.....</i>	<i>29</i>
OUTPUT DESCRIPTION AND EXAMPLES	29
<i>Workspace Outputs File</i>	<i>29</i>
<i>Data Visualization in GIS software</i>	<i>32</i>
KNOWN ISSUES	40
FINE-TUNED PARAMETERS	41
<i>Horizontal Curve.....</i>	<i>41</i>
<i>Vertical Curve.....</i>	<i>41</i>
APPENDIX C: LABEL RE-MAPPING GUI AND SOURCE	43
APPENDIX D: ADDITIONAL SUMMARY STATISTICS OF ACCURACY ASSESSMENT FOR MACHINE LEARNING-BASED CLASSIFICATION	55

APPENDIX A: QUICK USER GUIDE FOR GROUND FEATURE EXTRACTION SOFTWARE PROTOTYPE



User Manual for EZFeat V1.0

Support:

Michael Olsen, PhD., CEO
Michael.olsen@ezdatamd.com

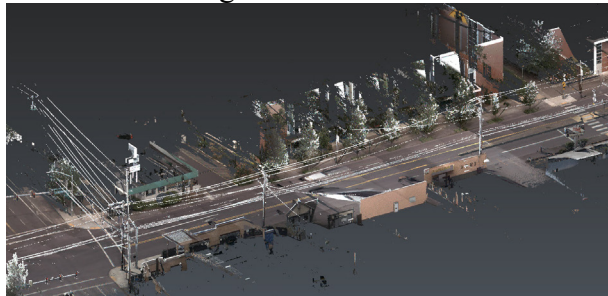


BACKGROUND

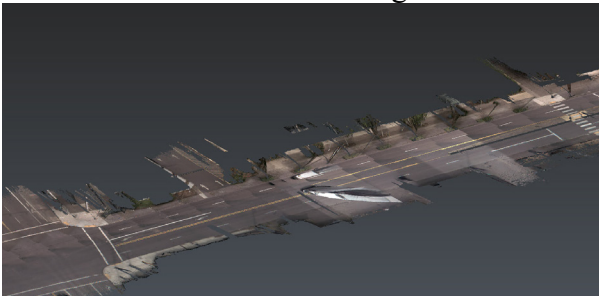
Welcome to *EZFeat*! *EZFeat* is a robust software solution designed for the automatic and comprehensive processing of point cloud data, including ground filtering and the detection of several road features including curb, road path, and marking. *EZFeat* has the following features:

- Automatic Ground Filtering: efficiently identifies and separates ground points from non-ground elements (e.g., buildings, vegetation, and other objects) within point cloud data, enabling clean and accurate terrain representation.
- Road Feature Detection: accurately identifies and extracts key road features, including curbs, road paths, and markings, providing precise data analysis for road environments.

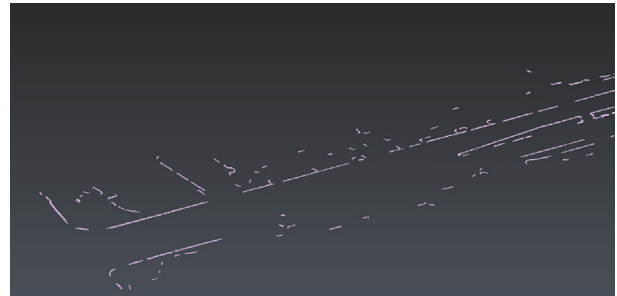
Original Point Cloud



Ground Filtering



Curb Detection



Road Path Detection



Road Marking Detection

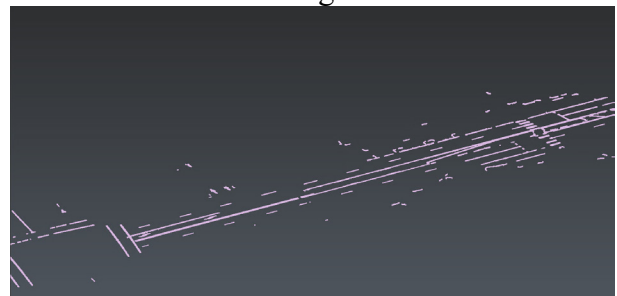


Figure A. 1 EZFeat Example Inputs and Outputs

SOFTWARE SETUP

EZFeat can be installed by unzipping and running the EZDataMD_EZFeat_GUI_Installer.exe. Then, select a destination folder for the installation. The default is C:\lidar\EZDataMD\.

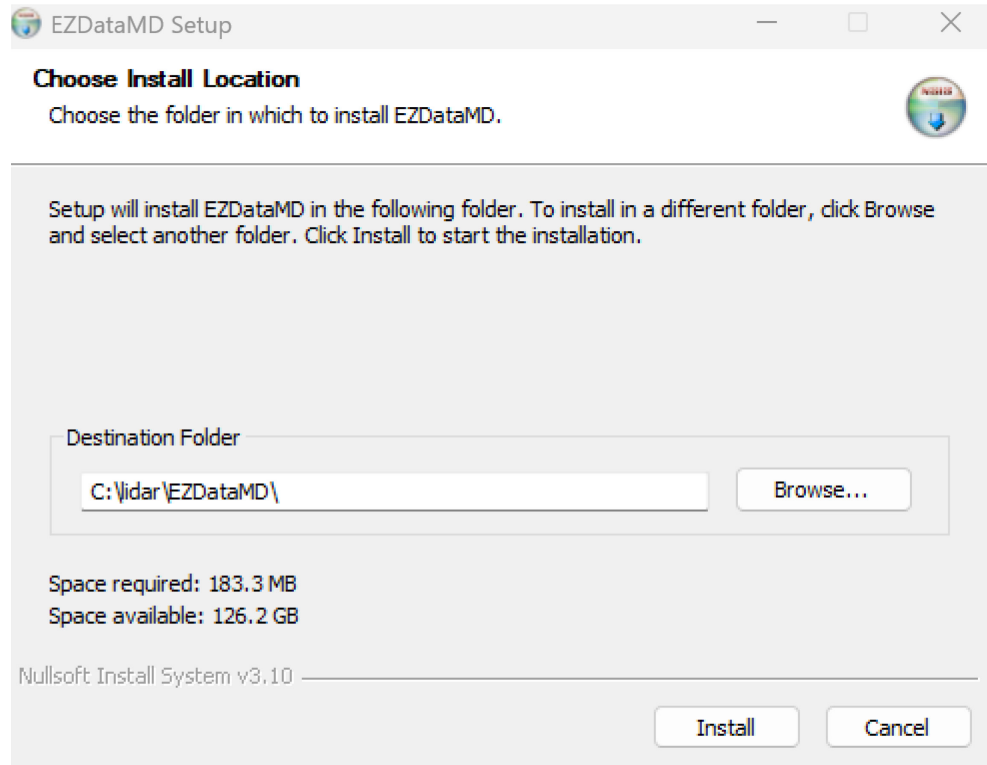


Figure A. 2 Installer GUI

Note:

- A current license file needs to be in the user's EZLicense folder or the same directory as the executable in order for *EZFeat* to run. Contact EZdataMD LLC (mail to: sales@ezdatamd.com) for an updated license, if needed.
- Tip: For optimal setup, locate the EZ_License.lst file in the extracted folder and open it using a text editor. Check if the path specified in the file matches the actual location of the user's license file. If there is a mismatch, user can either update the path in EZ_License.lst to reflect the correct location of the license file or move the license file to the location specified in EZ_License.lst.
- During uninstall, the uninstaller will remove all files within the folder in which the GUI is installed. So other files should not be stored in that directory.

BASIC PROGRAM SETUP

Starting EZFeat

Select EZFeat from the Start Menu (or use the desktop shortcut). You will need to provide node.js with permission for public and private networks to access the app.

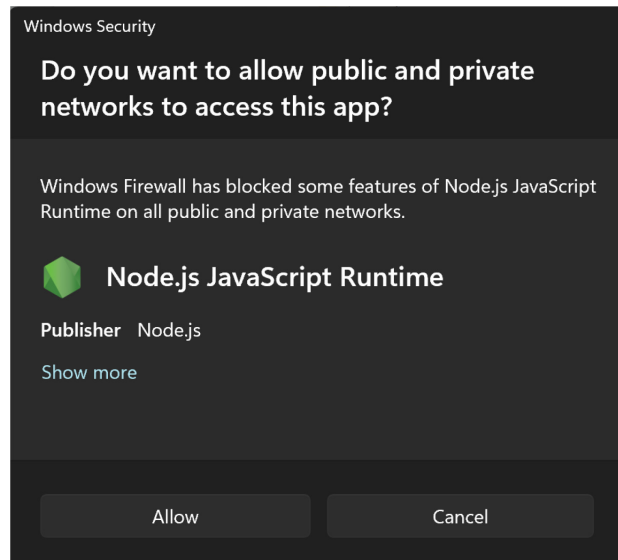


Figure A. 3 *Permission pop-up window.*

Upon granting permission, both a command prompt and GUI will load. The command prompt will provide status updates and runs in the background. The GUI is how the user interacts with the program to select workspaces and establish settings.

A Windows Command Prompt window titled "EZFeat_GUI". It displays the following output:

```
write the root path of the script to path.json: C:\Users\olsen\AppData\Local\Temp\_MEI180682
write the root path of the script to path.json: C:\Users\olsen\AppData\Local\Temp\_MEI180682\path.json
init the server program: C:\Users\olsen\AppData\Local\Temp\_MEI180682\voxelcompare-api.exe
wait for 5 seconds...
configuring taskprogresscheckcron
sqlite3 {
  Database: [Function: Database],
  Statement: [Function: Statement],
  Backup: [Function: Backup],
  OPEN_READONLY: 1,
  OPEN_READWRITE: 2,
  OPEN_CREATE: 4,
  OPEN_FULLMUTEX: 65536,
  OPEN_URI: 64,
  OPEN_SHARED_CACHE: 131072,
  OPEN_PRIVATE_CACHE: 262144,
  VERSION: '3.44.2',
  SOURCE_ID: '2023-11-24 11:41:44 ebead0e7230cd33bcec9f95d2183069565b9e709bf745c9b5db65cc0cbf92c0f',
  VERSION_NUMBER: 3044002,
  OK: 0,
  ERROR: 1,
  INTERNAL: 2,
  PERM: 3,
  ABORT: 4,
  BUSY: 5,
  LOCKED: 6,
  NOMEM: 7,
  READONLY: 8,
  INTERRUPT: 9,
  IOERR: 10,
```

Figure A. 4 Command Prompt Window with output diagnostic information

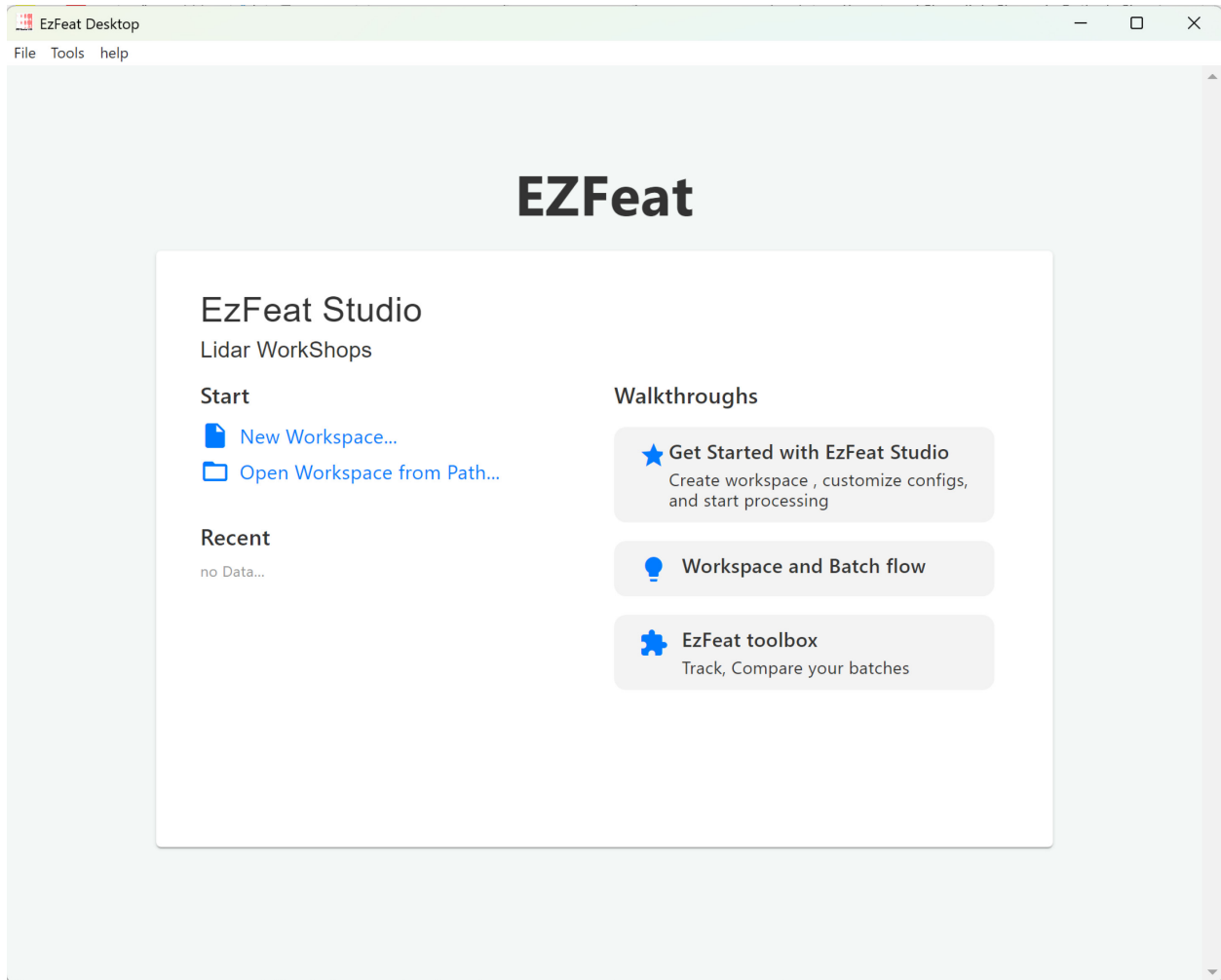


Figure A. 5 Main GUI of EZFeat.

Executables

The program contains two executables, described in the table below.

Executable	Description	Inputs	Outputs
voxel-demo.exe	This is the main executable to run <i>EZFeat</i> . Options that can be run from <i>EZFeat</i> are described in Section 3.1.	FILENAME.1 az	See Section 4.2.
voxelcompare-api.exe	This is a background executable that connects to the server. Run this first before running the main executable and leave it open while running EZFeat.	N/A	N/A

EZFeat modes

When a user provides a laz file to EZFeat, they can select from four different options of processing. The following table outlines the different processing options accessible through the software and the associated input and output files.

Option	Description	Inputs	Outputs
Ground filtering	The process of identifying and separating points that represent the ground surface from those that represent non-ground objects in point cloud data. This is based on a versatile ground filter based on multi-scale voxelization and smooth segment (Vo-SmoG) algorithm (Che et al. 2021a).	FILENAME.laz	FILENAME_grnd.laz
Curb detection	The process of identifying and extracting curbs from point cloud data. This is based on the algorithm of Che et al. (2023).	FILENAME_grnd.laz	FILENAME_grnd_curb.laz FILENAME_grnd_curb_ezft.laz
Road detection	The process of identifying and extracting road surfaces from point cloud data. The software uses the VROOM algorithm from Che et al. (2021b).	FILENAME_grnd.laz FILENAME_grnd_curb.laz	FILENAME_grnd_curb_road.laz FILENAME_grnd_curb_road_ezft.laz
Mark detection	The process of identifying and extracting road markings from point cloud data.	FILENAME_grnd.laz FILENAME_grnd_curb.laz FILENAME_grnd_curb_road.laz	FILENAME_grnd_curb_road_mark.laz FILENAME_grnd_curb_road_mark_ezft.laz

COMPONENTS OF A PROJECT

Input Files

EZFeat requires point cloud data in the LAS or LAZ compressed file format as input.

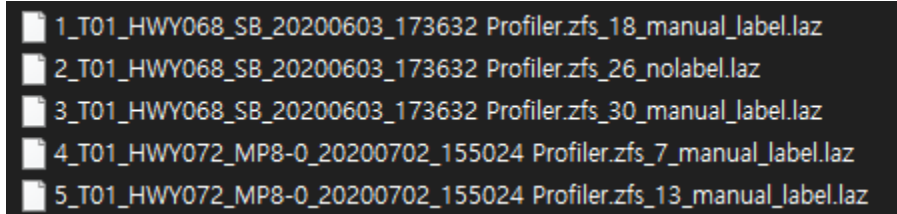


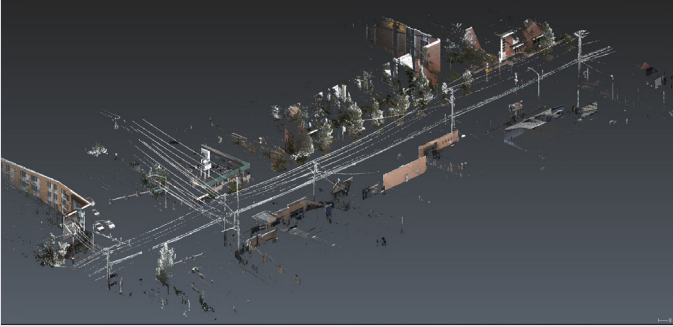
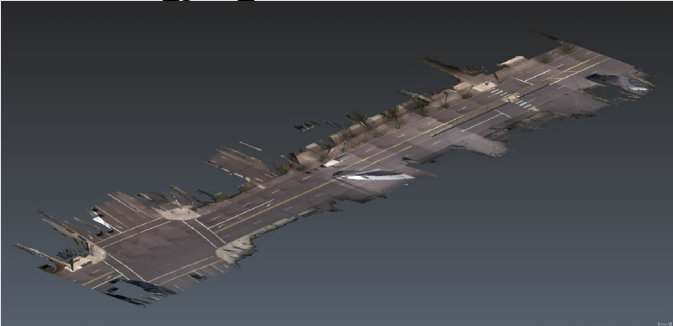
Figure A. 6 Output Files

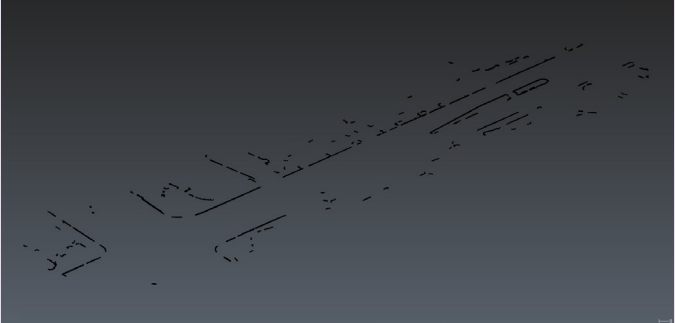
EZFeat generates multiple output files based on the selected options. Each output file contains a distinct group of point clouds categorized by the detected objects.

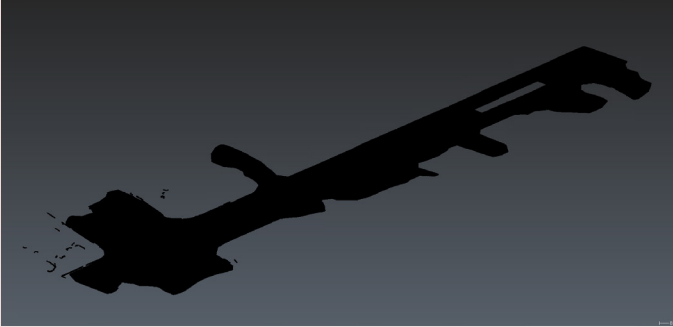
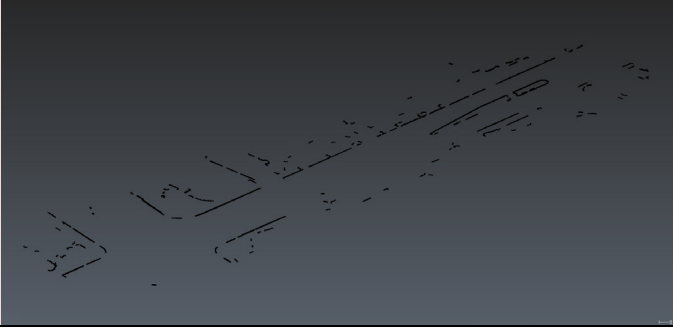
File	Description	Comments
__vosmog.txt (Ground filtering) __ezfeat_curbdetection.txt __ezfeat_roaddetection.txt __ezfeat_markingdetection.txt	These files are log files generated during the execution of each selected option. They contain detailed information about the operation of each module, including processing steps, results, and processing time.	N/A
__vosmog.Lst __ezfeat_curbdetection.Lst __ezfeat_curbdetection_ezft.lst __ezfeat_roaddetection.lst __ezfeat_roaddetection_ezft.lst __ezfeat_markingdetection.lst __ezfeat_markingdetection_ezft.lst	These files are a list of the created files resulting from each selected option.	These files ensure that each step uses the correct files from the previous steps to perform the next steps.
FILENAME_OPTION.laz	This file is the resulting point cloud according to the selected options. Each file contains sub-point cloud groups according to the options.	The output is determined by the selected option. For example, FILENAME_grnd_road_ezft.laz, FILENAME_grnd_curb_road_ezft.laz etc.
FILENAME_OPTION_ezft.laz	This file is a point cloud with objects separately extracted according to the options.	'_ezft' file contains mask layers of features extracted with the user-selected option.

If the user proceeds through all steps without skipping any, the output files and point cloud layers will consist of those shown in the table below. Some processing step creates two versions. One is the updated LAS/LAZ file with updated classifications for the points. The EZFT layers are simplified point clouds that just contain the primary object of interest (e.g., road surface, curb, pavement markings).

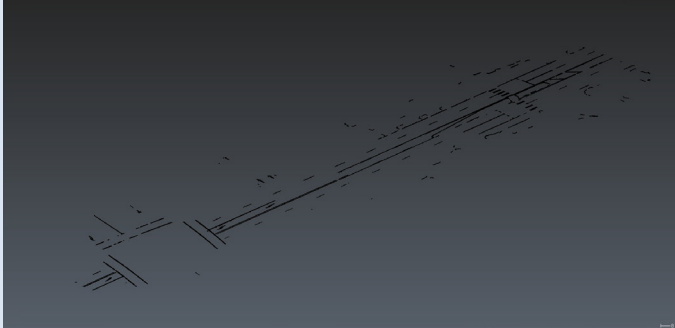
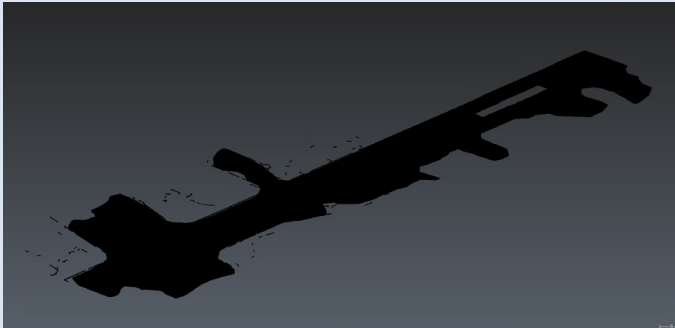
File	Output Point Cloud	Layers	Comment
01_Ground_Filtering	FILENAME_grnd.laz	FILENAME_grnd – Unclassified	Non ground points in the dataset that are labeled as unclassified.
		FILENAME_grnd – Ground	Ground filtered point cloud.
02_Curb_detection	FILENAME_grnd_curb.laz	FILENAME_grnd_curb – Unclassified	Non ground and non-curb points in the dataset such as buildings, vegetation, powerlines, and utility poles. Note that these objects are all tagged as “unclassified” and are not individually classified.

File	Output Point Cloud	Layers	Comment
			
		FILENAME_grnd_curb – Ground 	Ground filtered point cloud with curbs classified.
		FILENAME_grnd_curb – Created, never classified	Points that were not able to be classified as either ground or non-ground.
	FILENAME_grnd_curb_ezft.laz	FILENAME_grnd_curb_ezft – Created, never classified	Mask corresponding to the extracted curbs, with some interpolation implemented to preserve continuity of the curbs.

File	Output Point Cloud	Layers	Comment
			
03_Path_Detection	FILENAME_grnd_curb_road.laz	FILENAME_grnd_curb_road – Unclassified	Points that still remain as unclassified in the point cloud.
		FILENAME_grnd_curb_road – Ground	General ground points excluding the road surface.
		FILENAME_grnd_curb_road – Road Surface	Extracted road surface.

File	Output Point Cloud	Layers	Comment
			
		FILENAME_grnd_curb_road – Created, never classified	Points that were not able to be classified.
	FILENAME_grnd_curb_road_ezft.laz	FILENAME_grnd_curb_road_ezft – Road Surface 	Simplified point cloud with a mask corresponding to the extracted road surface.
		FILENAME_grnd_curb_road_ezft – Created, never classified 	Simplified point cloud with a mask corresponding to the extracted curbs.

File	Output Point Cloud	Layers	Comment
04_Mark_Detection	FILENAME_grnd_curb_road_mark.laz	FILENAME_grnd_curb_road_mark – Unclassified	Points that still remain as unclassified in the point cloud.
		FILENAME_grnd_curb_road_mark – Ground	Point cloud excluding road surface from ground filtered point cloud.
		FILENAME_grnd_curb_road_mark – Road Surface	Extracted road surface.

File	Output Point Cloud	Layers	Comment
	FILENAME_grnd_curb_road_mark_ezft.laz	FILENAME_grnd_curb_road_mark – Created, never classified	
		FILENAME_grnd_curb_road_mark_ezft – Created, never classified 	Simplified point cloud with a mask corresponding to the extracted road markings.
		FILENAME_grnd_curb_road_mark_ezft – Road Surface 	Simplified point cloud with a mask corresponding to the extracted road surface.

GRAPHICAL USER INTERFACE (GUI)

Interface and Usage

This section describes the features and detailed usage of *EZFeat*'s main GUI. If the user follow the numbered sequence in the Figures, user will be able to easily perform ground filtering and other detection tasks.

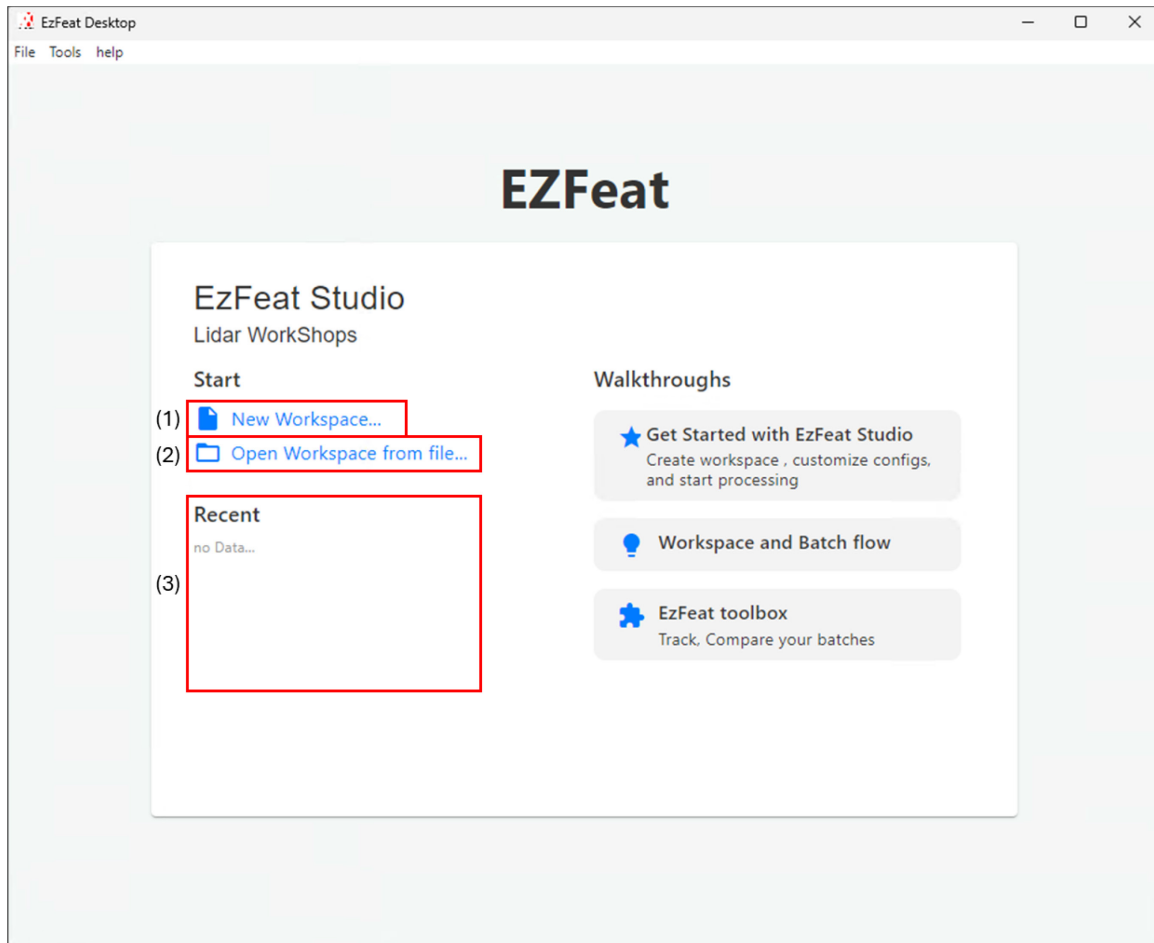


Figure A. 7 Screenshot of the GUI

Below are descriptions of each element in the GUI.

1. Create a new workspace. All outputs will be saved here.
2. Specify a workspace from one already created. All outputs will be saved here.
3. The user can check the recently used workspaces in this box.

Upon setting the workspace, click *Next* to open the *Ground Filtering* tab.

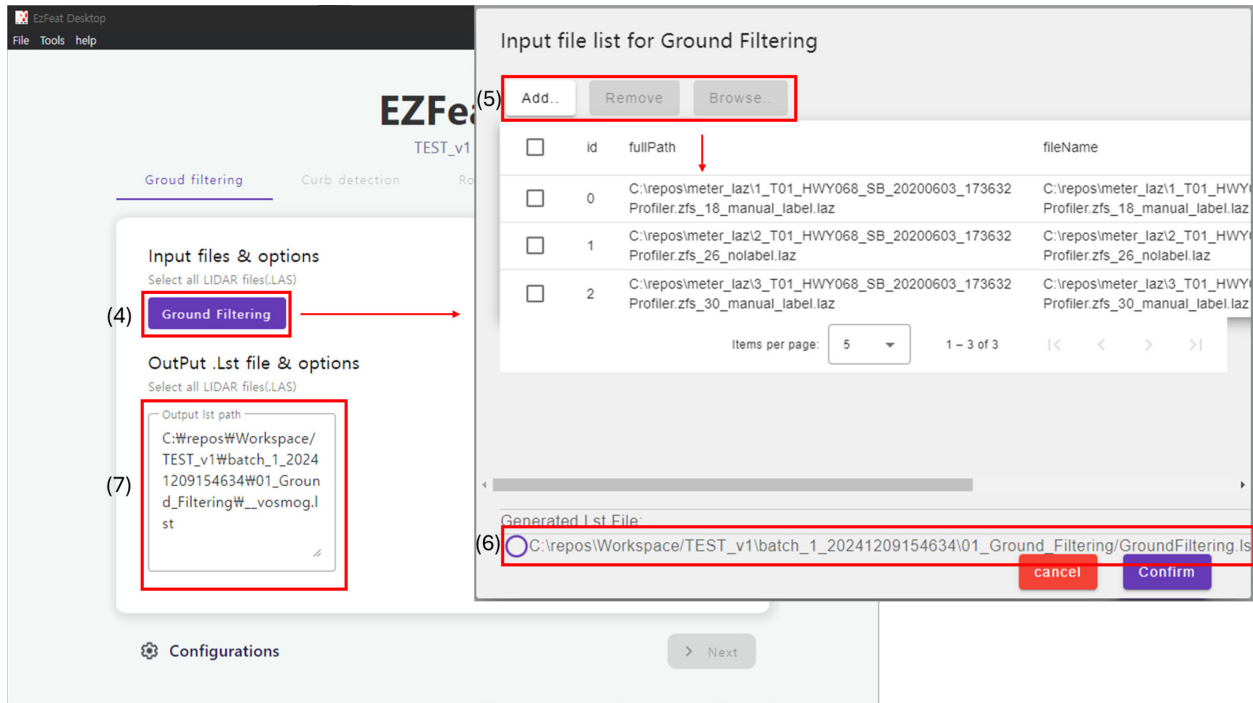


Figure A. 8 Screenshot of the Ground filter tab and associated file browser

The following options are available through this tab in the GUI:

4. This is where users set the path for the input files.
5. Click 'Add' to specify the point cloud data users want to process.
6. The lst file will be generated directly, it will work as a trigger for the whole workflow.

Here users can check the path of the output lst file. EZFeat automatically creates subfolders to store the results. The names of the subfolders are as shown in the screenshot below. This completes the settings for 'Ground filtering'. Click 'Next' button and move to next step.

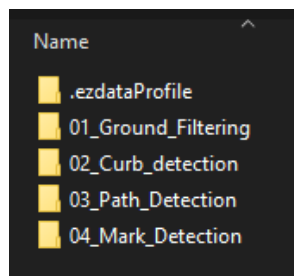


Figure A. 9 Output folder structure that are automatically created

Tip: When one task is finished and moves on to the next task, EZFeat automatically creates the next task file and specifies the path. In the screenshot above, it is 'batch_1_', but in the next task, EZFeat automatically creates a subfolder in 'batch_2_'.

Click 'Confirm' button and move to the next step.

Tip: When performing a new task, please check (5) and (6) again, even if it uses the same data files. The user can check which data files are loaded and performed.

The next step is 'Curb detection'.

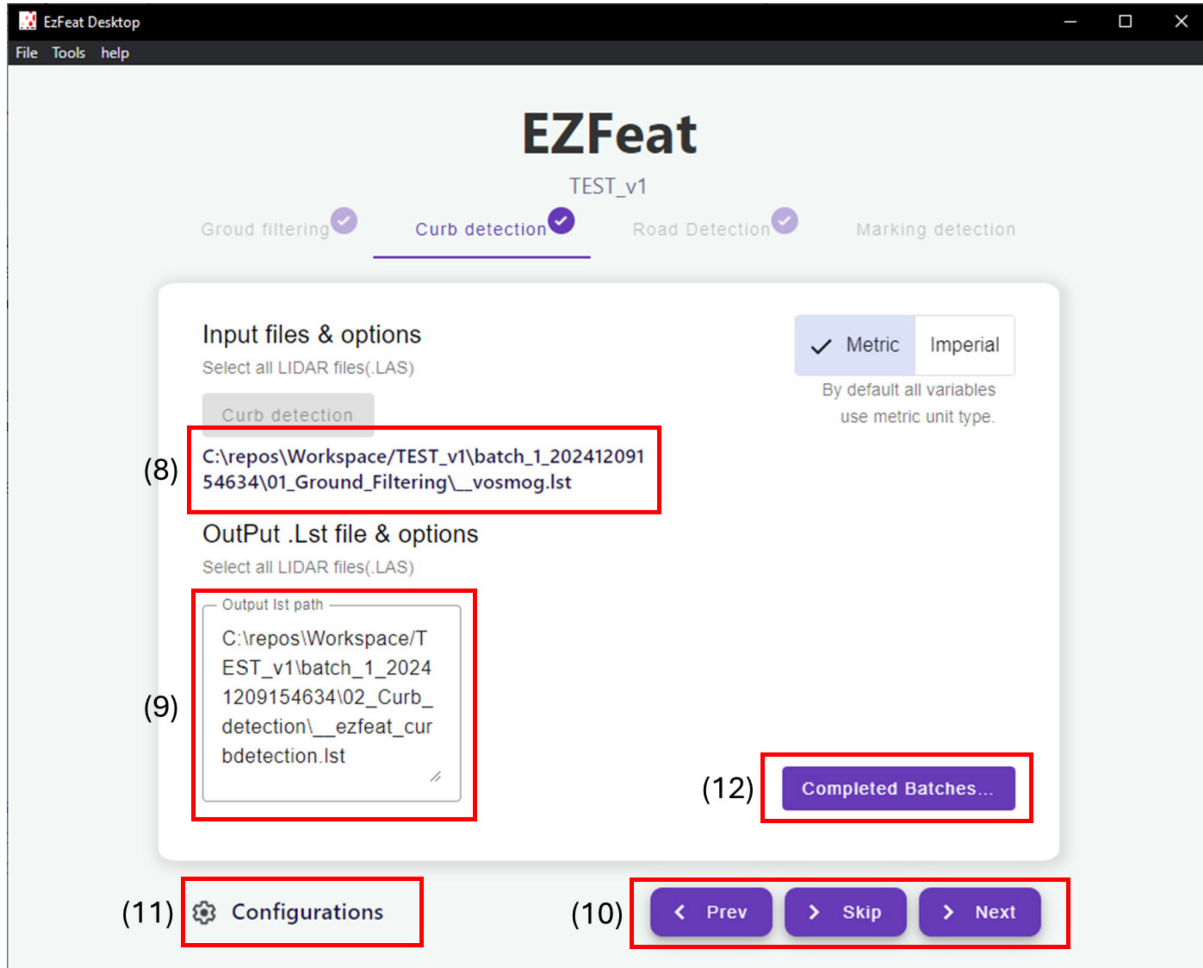


Figure A. 10 Curb Detection Tab

7. The next step is 'Curb detection'. Here users can check the path of the input file to be used in 'Curb detection'.
8. Here users can check the path of the output file after performing the 'Curb detection'.
9. The user can go back to previous steps, skip the Curb detection task, or continue to the next step by clicking 'Prev', 'Skip', and 'Next'.
10. In 'Configurations', users can manage workspaces and batches. The user can also set parameters through the configurations step (see the end of this section for details).
11. 'Completed Batches' allows users to copy configurations and data from a previously completed batch.

'Road detection' and 'Marking detection' also follow the same procedure.

Tip: If the user 'Skip' a task and click 'Next' on the next task, a pop-up window similar to the screenshot below will appear. Since the user skipped the previous task, the user needs to specify the input file to be used in the next task. There are two options: 1) use the output of the last step that has not been skipped and 2) specify an input file arbitrarily. Regardless of which the user chooses, it is recommended to double-check the paths in (8) and (9).

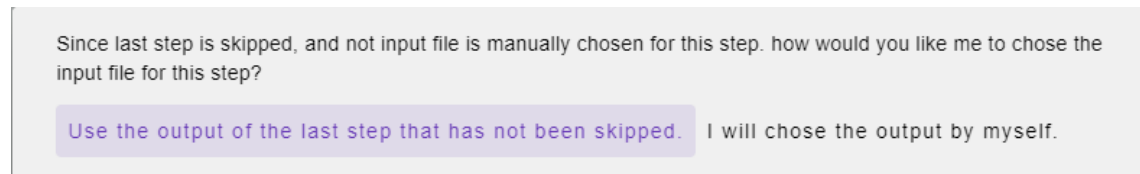


Figure A. 11 Screenshot of the pop-up

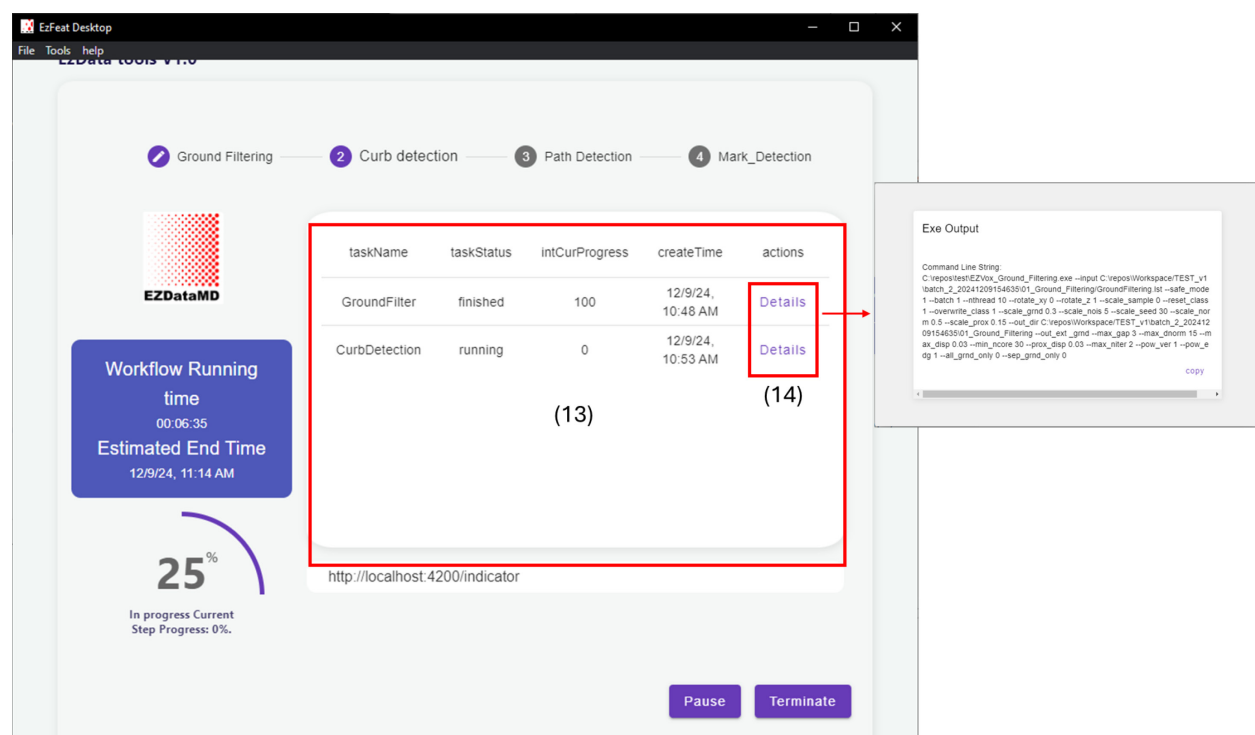


Figure A. 12 Processing tab where the user can monitor the status of processing

12. After setting all parameters for each step, the processing tab will show.
13. On this page, tasks will run automatically, and progress will be shown in the center box.
14. 'Details' show the output of the executable file and command string used.

Tip: The output log files of each step will be generated after it is done. They contain detailed information about the operation of each module, including processing steps, results, and processing time.

Upon completion of the processing tasks, the GUI will update to show the final information.

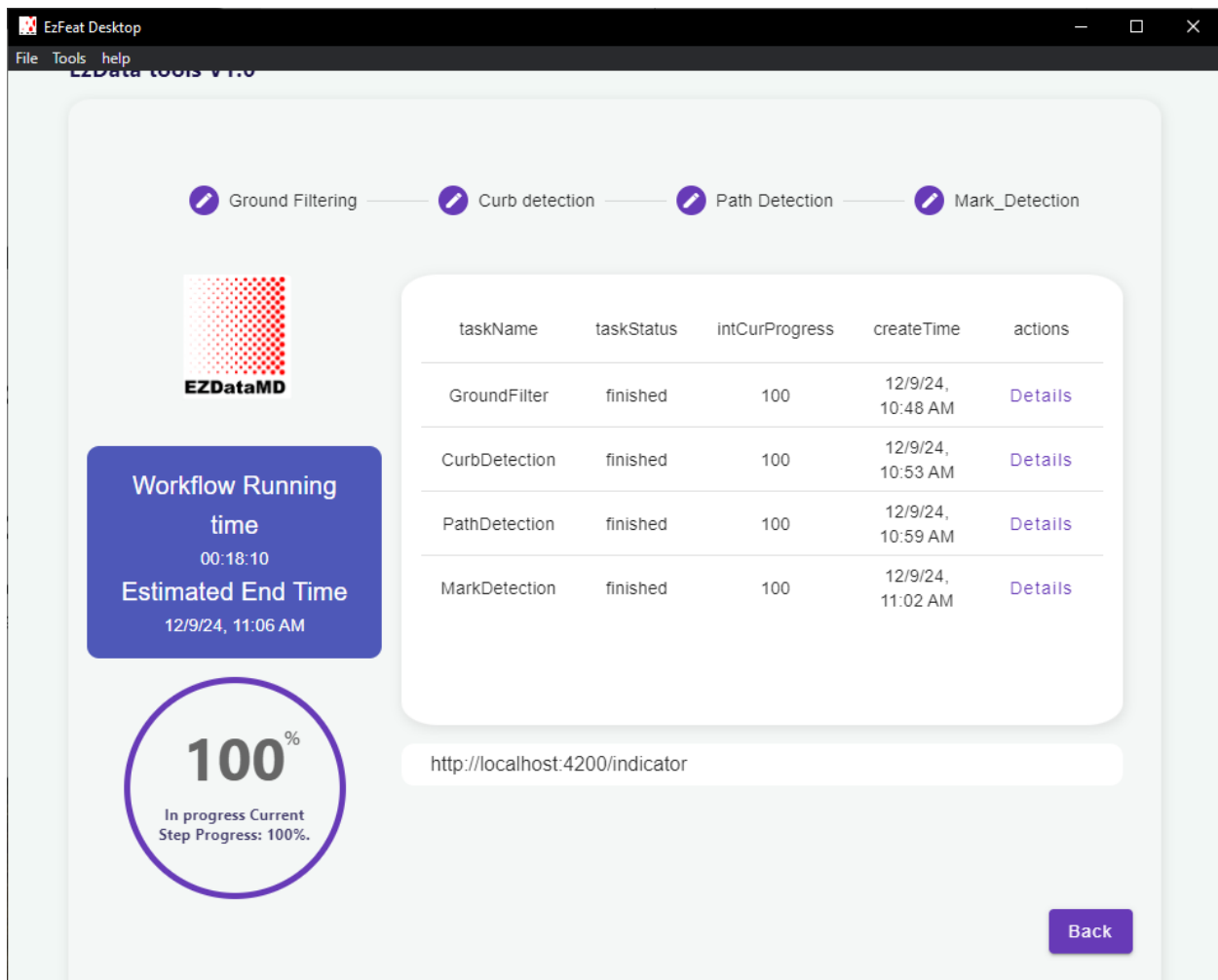
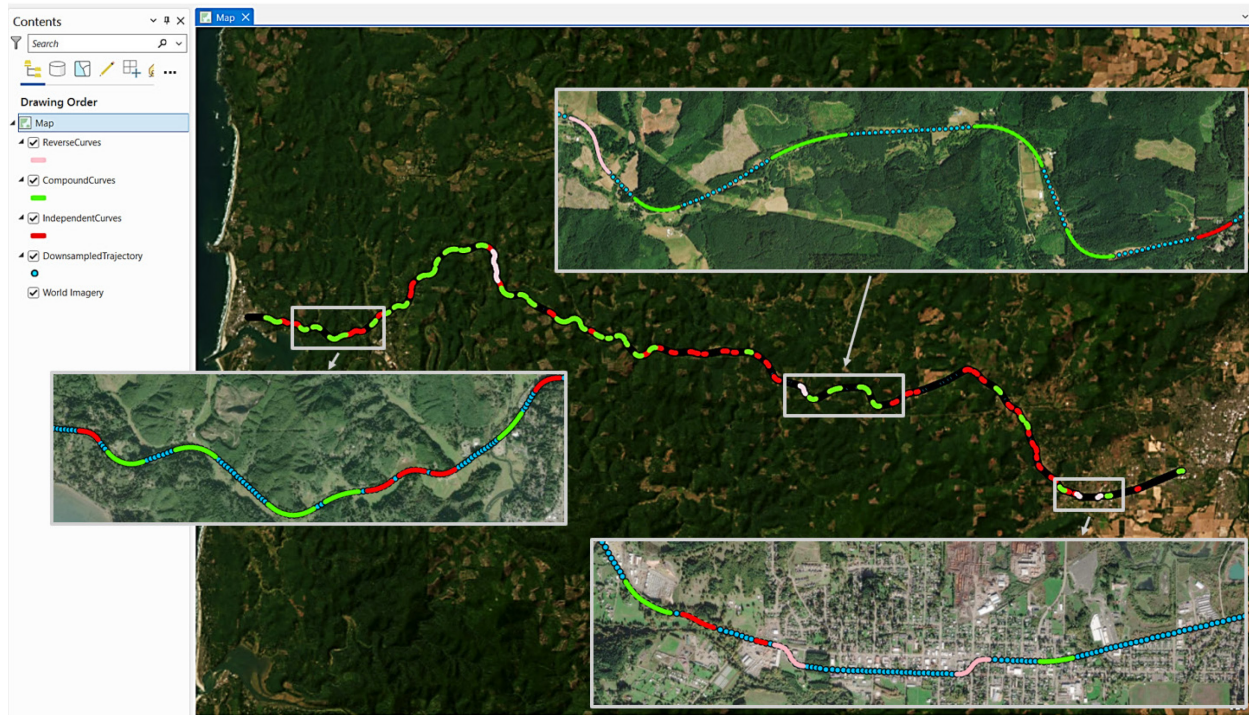


Figure A. 13 GUI showing status after completion of processing tasks

CONTRIBUTIONS

EZFeat uses tools developed by EZDataMD that are licensed from Oregon State University. Dr. Ezra Che from Oregon State University/EZDataMD is the lead developer of the tools executed in EZFeat. Dazsdigital, Inc. created the base code for the GUI. Dr. Dae Kun Kang conducted testing of the GUI and developed the initial draft of the User Manual. Dr. Michael Olsen has provided oversight to the code development, testing, and product development.

APPENDIX B: QUICK USER GUIDE FOR GIS TOOLBOX PROTOTYPE



Developed by
Sahar Kamalou, Ezra Che
Oregon State University

02/13/2025

INSTALLATION

ArcGIS Pro Installation

This toolbox requires ArcGIS Pro with arcpy library to be installed on the same system for running geospatial analysis.

ArcGIS Pro Python Environment

ArcGIS Pro comes with its own Python environment (a conda environment), which is located within the ArcGIS Pro installation directory. This environment contains the arcpy library but may not include some of the other necessary libraries by default. To install the required libraries, either the ArcGIS Pro Python Package Manager or install them via a terminal or command prompt using the pip install command, depending on the preferred Python development environment. Cloning the arcgispro-py3 environment in ArcGIS Pro allows users to create a copy of the default Python environment, which can be then modified without affecting the original environment. To Clone the arcgispro-py3 environment follow the steps below.

ArcGIS Pro > Package Manager > Activate Environment > 

Step 1: Next to the arcgispro-py3 environment, click the Clone button.

Step 2: In the dialog box, the directory and name of the cloned environment can be defined.

Step 3: Click OK to start the cloning process. This might take a few minutes, depending on the size of the environment and installed packages. Once the cloning is complete, the new environment will appear in the list. Click on the cloned environment to activate it.

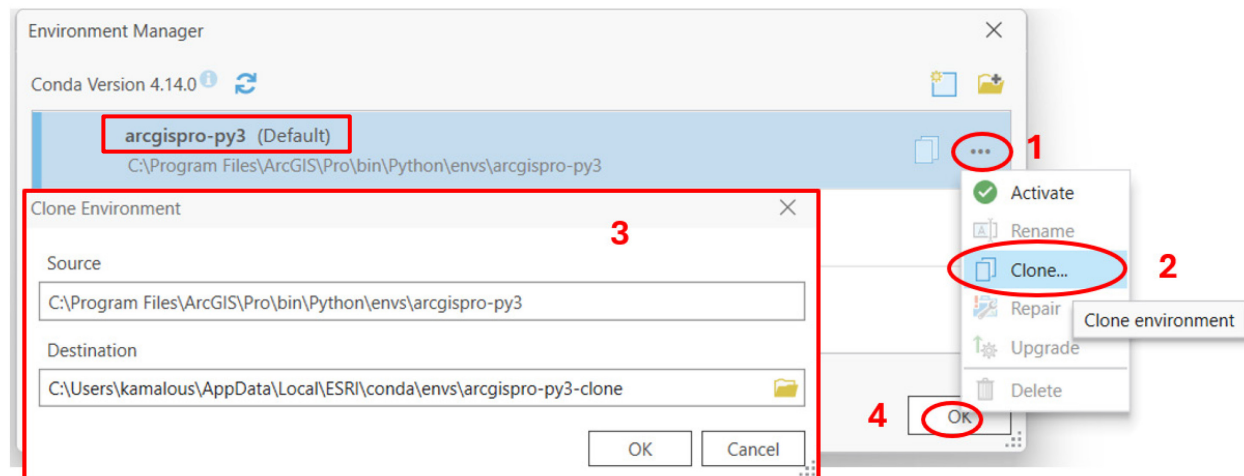


Figure B. 1 Creating a clone environment in ArcGIS Pro.

Configure Python Libraries

To run the provided ArcGIS Pro toolboxes, the following Python libraries are required to be installed. It is better for user install libraries through ESRI channel as much as possible to ensure the compatibility. To install the following libraries, in Package Manager, select cloned environment in Active Environment, then install desired libraries from Add Packages.

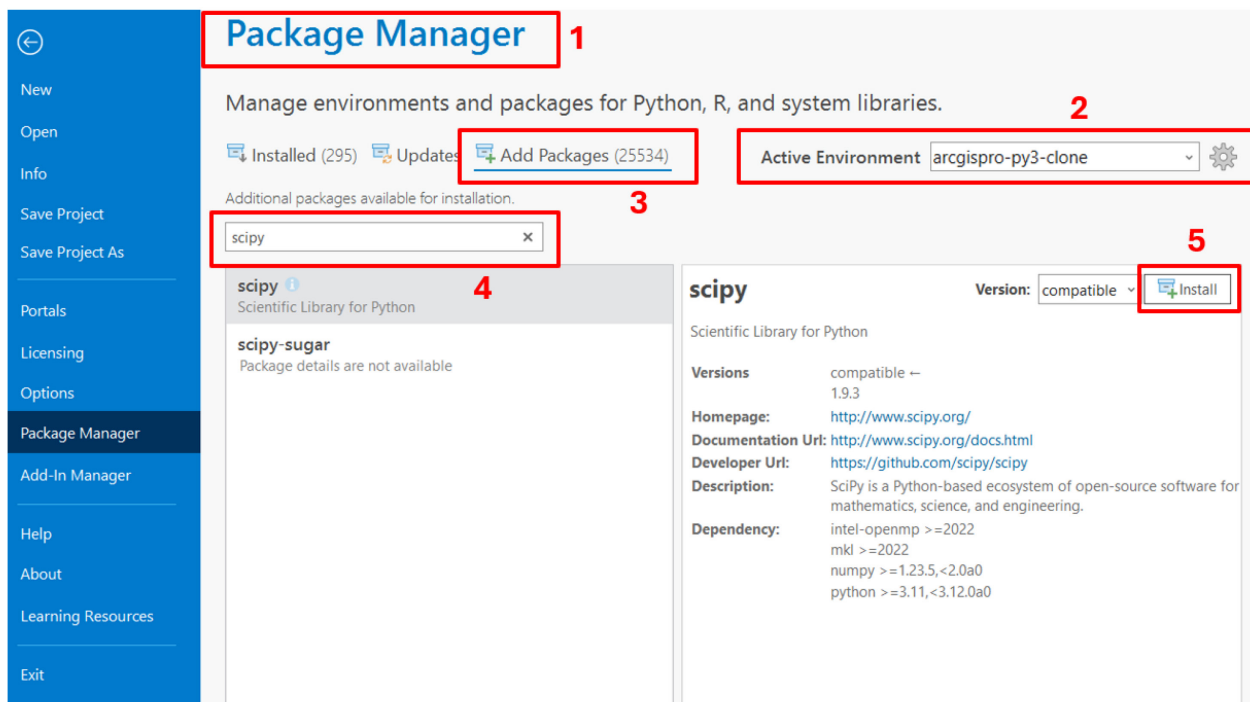


Figure B. 2 CrossSection Tool interface in ArcGIS Pro

Some libraries such as “laszip” cannot be installed in the ArcGIS Pro Python Package Manager. In this case, manual installation is needed. To manually install third-party libraries (in this case, laszip), open the “Python 3 Command Prompt”. If it does not exist in the start menu, locate the **proenv.bat** in the ArcGIS folder, and here is an example:

```
<install>\ArcGIS\Server\framework\runtime\ArcGIS\bin\Python\Scripts
```

Make sure the active environment is the cloned one created following the previous steps. More details can be found here: <https://enterprise.arcgis.com/en/server/11.2/publish-services/windows/deploying-custom-python-packages.htm>. Then for installing laszip library, simply type in “pip install laszip” to install.

```

C:\WINDOWS\system32\cmd. x  +  -  x
(arcgispro-py3-clone) C:\Users\chee\AppData\Local\ESRI\conda\envs\arcgispro-py3-clone>pip install laszip
Active environment                                Install laszip

```

Figure B. 3 Script for installing laszip to ArcGIS Pro environment

Alternatively, if there is another preferred Python development environment, such as PyCharm. Then, navigate to the Python Interpreter settings, click Add Interpreter, and specify the path to the cloned environment's python.exe as the base interpreter.

Table B. 1 Dependencies for the ArcGIS toolbox developed in this project.

Library	Description	pip install	ArcGIS Pro Python Environment install
scipy	This library is used for scientific computing and is not included in the default ArcGIS Pro environment.		✓
pandas	This library is essential for data manipulation and analysis.		✓
matplotlib	This library is used for creating visualizations and plots.		✓
shapely	This library handles and manipulates geometric objects like points, lines, and polygons.		✓
laspy	Required for reading and writing LAS files, which are commonly used in LiDAR data processing.		✓
laszip	LASzip is required if LAZ format is used.	✓	
geopandas	This library extends pandas to handle geospatial data and is essential for analyzing and managing geospatial datasets in tabular formats.		✓
lxml	The lxml library is used for parsing, creating, and manipulating XML and HTML documents.		✓

ADDING AND RUNNING TOOLBOX

Horizontal Toolbox

Once the required software and libraries have been installed, the toolbox (Horizontal.pyt file in Horizontal_Toolbox folder) can be dragged and dropped to the Catalog Pane in ArcGIS Pro.

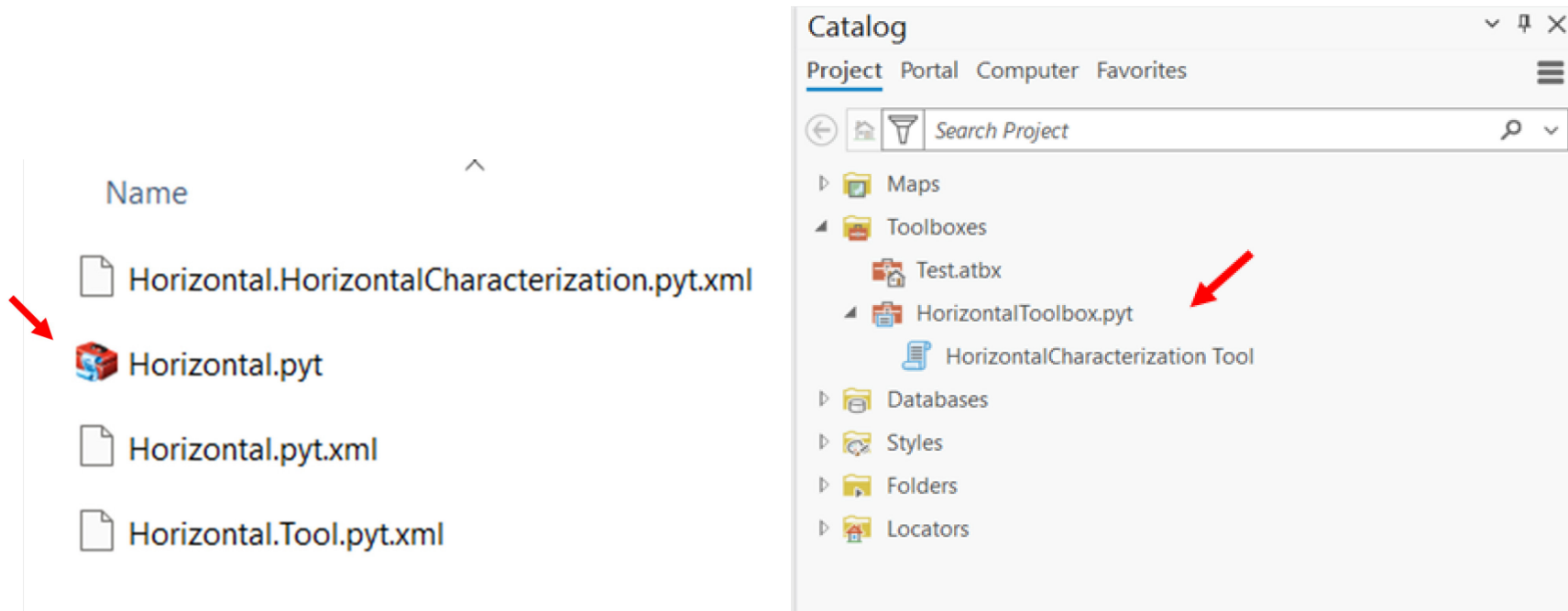


Figure B. 4 Adding toolbox to ArcGIS Pro Catalog

Then, the tool can be open for users to provide input data and other parameters/options.

Geoprocessing

HorizontalCharacterization Tool

Parameters Environments

* InputFile

MinimumDistance 100

GPSTime 2.5

Lower_CurvatureThreshold 0.15

Upper_CurvatureThreshold 3.5

* Workspace

* CoordinateSystem

* GeodataBase

☐ VisualResult

Run

Figure B. 5 HorizontalCharacterization Tool interface in ArcGIS Pro

For the first run, it is recommended to use the default values followed by multiple iterations adjusting the parameters to achieve the desired results. The following table provides guidance on the typical range of each parameter. For each parameter, the user can begin by running the toolbox with the default values and then adjust them within the provided typical range (Figure B. 2) to achieve the desired results. Recommended parameters for the sample datasets are provided in the Appendix.

Table B. 2 Overview of input parameters

Parameter	Description	Typical Range	Default Value	Unit
InputFile	The trajectory file in .txt format that will be used as input.	-	-	-
MinimumDistance	This field takes the desired minimum distance between trajectory points after down sampling.	50-300	100	Foot
GPSTime	This field specifies the desired time interval between trajectory points for down sampling. The larger the value, the greater the distance between Trajectory points after downsampling.	1-3	2.5	Seconds
Lower CurvatureThreshold	By fitting a circle to each trajectory point in horizontal plane, and using the inverse of its radius, a curvature value was assigned to each point. Points are then segmented by applying a threshold on their curvature. The smaller this value, the more identified the horizontal curve is.	0.1-0.9	0.15	Foot ⁻¹
Upper CurvatureThreshold	The larger this value, the less accurately roundabouts, turns, and exits are identified.	3-15	3.5	Foot ⁻¹
Workspace	The location for storing output files will be specified here. The user should create a folder on their local system and provide its path in this field.	-	-	-
CoordinateSystem	This will be the coordinate system of the input and output.	-	-	-
GeodataBase	User can create a new geodata base or use the created geodatabase in catalog and enter its path here.	-	-	-
Visual Results	By checking this box visual results will appear in the map.			

After entering all the required parameters using the following table, press the 'Run' button to start the processing. Each toolbox may take varying amounts of time to execute. Enabling the Visual Result option could significantly increase the program's runtime. With Visual Result enabled, the analysis typically takes several minutes to complete, whereas without showing the visual results, the analysis can be completed in a few seconds. Once the analysis is completed, all output results will be saved as a number of CSV files in the workspace folder. Additionally, if the Visual Results option is enabled, some map layers will be created and stored in the geodatabase to help users visualize the results.

Vertical Toolbox

Similar to the horizontal curve toolbox, the vertical toolbox (vertical.pyt in Vertical_Toolbox folder) can be added to the catalog in a similar way (drag-and-drop). The user interface and

suggested parameters are listed below. The visual results of the vertical toolbox include an elevation profile with the segmentation and identification results from the mobile lidar trajectory.

Geoprocessing

VerticalCharacterization Tool

Parameters Environments

* InputFile

MinimumDistance 100

GPSTime 2.5

Curvature_Threshold 0.3

* Workspace

Window_Size 5

* CoordinateSystem

* GeodataBase

☐ VisualResult

Run

Figure B. 6 VerticalCharacterization Tool interface in ArcGIS Pro

Table B. 3 Overview of input parameters

Parameter	Description	Typical Range	Default Value	Unit
InputFile	The trajectory file in .txt (Leica and RIEGL) format that will be used as input.	-	-	-
MinimumDistance	This field takes the desired minimum distance between trajectory points after down sampling.	50-300	100	Foot
GPSTime	This field specifies the desired time interval between trajectory points for down sampling. The larger the value, the greater the distance between Trajectory points after downsampling.	1-3	2.5	Seconds
Curvature_Threshold	By fitting a circle to each trajectory point in vertical plane and using the inverse of its radius, a curvature value was assigned to each point. Points are then segmented by applying a threshold on their curvature. The smaller this value, the more identified the vertical curve is.	0.1-0.7	0.3	Foot ⁻¹
Window_Size	In addition to curvature slope of each point has been used for segmentation. Window_Size is Size of searching window for slope calculation.	2-5	5	-
Workspace	The location for storing output files will be specified here. The user should create a folder on their local system and provide its path in this field.	-	-	-
CoordinateSystem	This will be the coordinate system of the input and output (Feet Intl).	-	-	-
GeodataBase	User can create a new geodata base or use the created geodatabase in catalog and enter its path here.	-	-	-
Visual Results	By checking this box visual results will appear in the map.			

Cross section Toolbox

Click on the Cross Section script to open it, then enter the required input parameters, and click the 'Run' button. Note that the current version does not open parameter settings to the users for now.

The screenshot shows the 'CrossSection Tool' interface within the 'Geoprocessing' window. The 'Parameters' tab is selected, displaying the following fields:

- * InputFile**: A text input field with a folder icon to its right.
- * Workspace**: A text input field with a folder icon to its right.
- * CoordinateSystem**: A dropdown menu with a globe icon to its right.
- * PointCloud**: A text input field with a folder icon to its right.
- * HorizontalCurveIdentification**: A text input field with a folder icon to its right.
- Geodatabase**: A text input field containing 'CrossSections.gdb' with a folder icon to its right.
- VisualResult**: A checkbox that is currently unchecked.

At the bottom right, there is a 'Run' button with a play icon and a dropdown arrow.

Figure B. 7 CrossSection Tool interface in ArcGIS Pro

Table B. 4 Overview of input parameters

Parameter	Description	Unit
InputFile	The trajectory file in .txt (Leica and RIEGL) format that will be used as input.	-
PointCloud	Detected road point cloud (.laz) should be enter in this field.	-
HorizontalCurve Identification	In this field, the user should enter the Filtered_Curve_Segments.csv file, which is one of the outputs from the Horizontal Characterization Toolbox. This file helps assign the ID of curves or straight sections to each cross-section line.	-
Workspace	The location for storing output files will be specified here. The user should create a folder on their local system and provide its path in this field.	-
CoordinateSystem	This will be the coordinate system of the input and output.	-
GeodataBase	User can create a new geodata base or use the created geodatabase in catalog and enter its path here.	-
Visual Results	By checking this box visual results will appear on the map.	-

Visualization Toolbox

The visualization tool is developed to visualize the results generated by three separate toolboxes (horizontal, vertical, cross section). The inputs for this toolbox are the outputs of those three toolboxes, as detailed in the table below. Once the inputs are loaded and the toolbox is executed, the visual results will appear on the map and be stored in the designated geodatabase. Each of the three toolboxes also offers an option to visualize their individual results after the analysis. It is worth noting that enabling this visualization will increase the processing time. To optimize the performance, especially when batch processing is needed, it is highly recommended running tools with visualization disabled. Then this tool can be used for visualization.

OUTPUT DESCRIPTION AND EXAMPLES

Workspace Outputs File

After running each toolbox, the results will be stored as .csv files in the specified Workspace. The table below lists the results containing the desired information. The files listed below will also be stored in the user-defined Geodatabase.

Table B. 5 Overview of output files.

Toolbox	Output	Description
Horizontal	Downsampled_Trajectory.csv	This file contains X, Y and Z coordinates (ft) of down sampled trajectory points.
	Roundabout_Turns_segments.csv	This file contains X, Y and Z coordinates (ft) of points which are located on roundabout, turns, and exists along the road. Each of them can be identified by its own unique label. The absence of this CSV file in the workspace after the processing indicates that, based on the road characteristics or selected parameters, no roundabouts, turns, or driveways have been identified.
	Independent_Curves.csv	It contains X, Y and Z coordinates (ft) of points which are located on independent horizontal curves along the road. Each curve is identified by its unique label.
	ReverseCurves.csv	This file contains X, Y and Z coordinates (ft) of points which are located on horizontal reverse curves. Each curve is identified by its own unique Curve ID. Since a reverse curve is formed from two curve parts, labels indicate which points are in each part.
	CompoundCurves.csv	This file contains X, Y and Z coordinates (ft) of points which are located on horizontal compound curves. Each curve is identified by its own unique Curve ID. Since a compound curve is formed from two or more adjoining curve parts curving in the same direction, labels indicate which points are in each part.
	Characteristics.csv	It includes horizontal curve characteristics such as the coordinates of the curve center, start point, and end point (X and Y in meters), curve radius (m), curve length (m), and deflection angle (degrees). It also contains Curve Type and Curve ID. These characteristics have been calculated and recorded for independent curves, as well as for each curve part in compound and reverse curves.

Toolbox	Output	Description
Vertical	Downsampled_Trajectory.csv	This file contains the X, Y, and Z coordinates (ft) of down-sampled trajectory points. Plotting the Z coordinates with their indices can render a vertical profile of the road segment.
	Updated_Curves.csv	It contains X, Y and Z coordinate (ft) of points which are located on vertical curves along the road. Each curve is identified by its own unique Label.
	Curve_Characteristics.csv	It includes vertical curve characteristics such as the coordinates of the start point, and end point (X and Y in meter), and curve length (m). These characteristics have been calculated and recorded for each curve.
Cross Section	CrossSection_Characteristics.csv	This file contains XYZ coordinates of points along each cross section.
	TabelResults.csv	This file includes characteristics such as cross slope for each cross section. Locations where cross sections have “Wide” in their label name are likely associated with intersections, turns, or driveways.

Data Visualization in GIS software

Horizontal Curve identification

Areas featuring horizontal independent, compound, and reverse curves are displayed on the map using polylines in different colors. Roundabouts, turns, exits, and driveways are indicated by yellow points. Blue points represent down-sampled trajectory points that are visible on straight sections of the road.

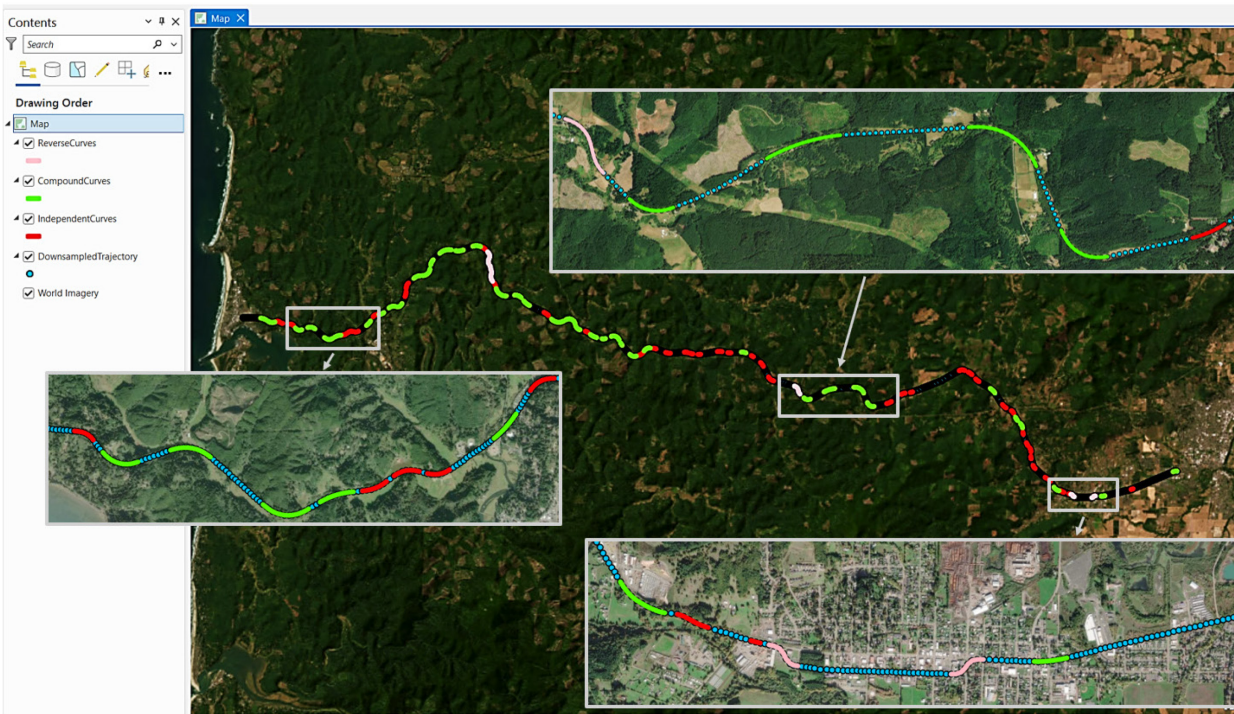


Figure B. 8 Curve identification results of Horizontal toolbox for RIEGL_export_Mission dataset.

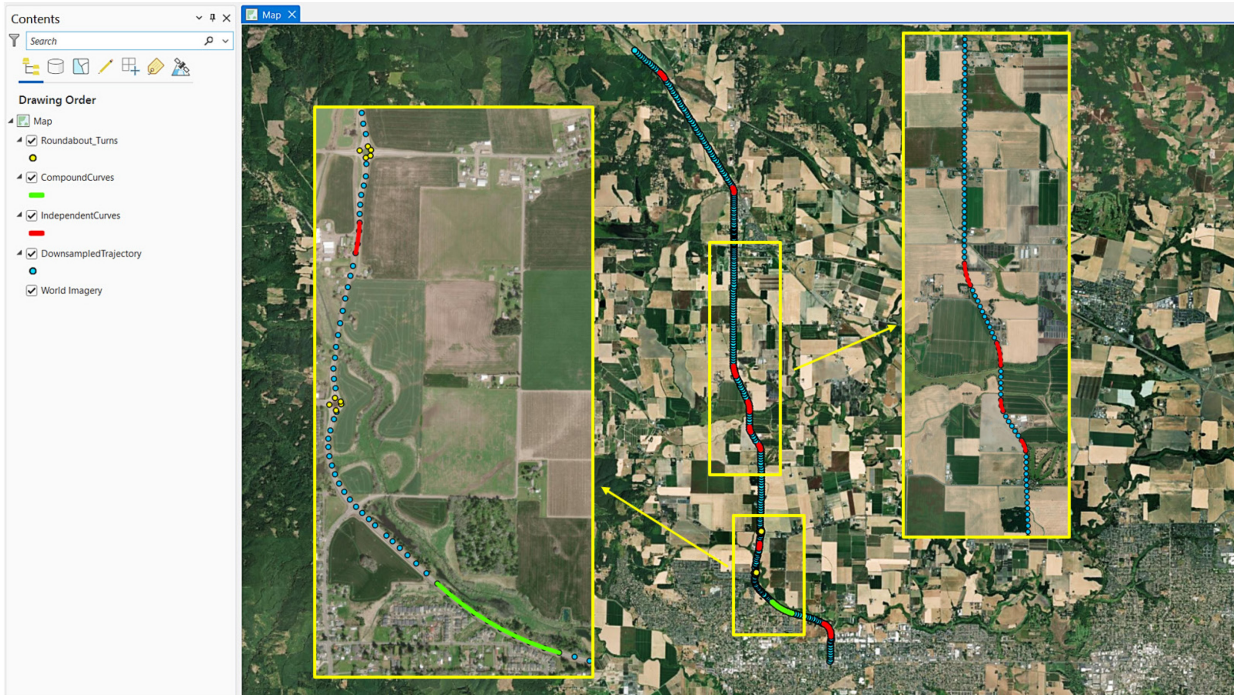


Figure B. 9 Curve identification results of Horizontal toolbox for T01_HWY072_MP8-0 dataset.

Horizontal Curve characterization

A pop-up table displaying the calculated characteristics of horizontal curves will appear on the map when user clicks on the curve polylines. These characteristics include coordinates of the curve center, start point, and end point (X and Y in meters), curve radius (m), curve length (m), and deflection angle (degrees), which are calculated for each unique curve. If the Curve Type is compound or reverse, the window will have two pages. Each page includes details for each part of the curve.

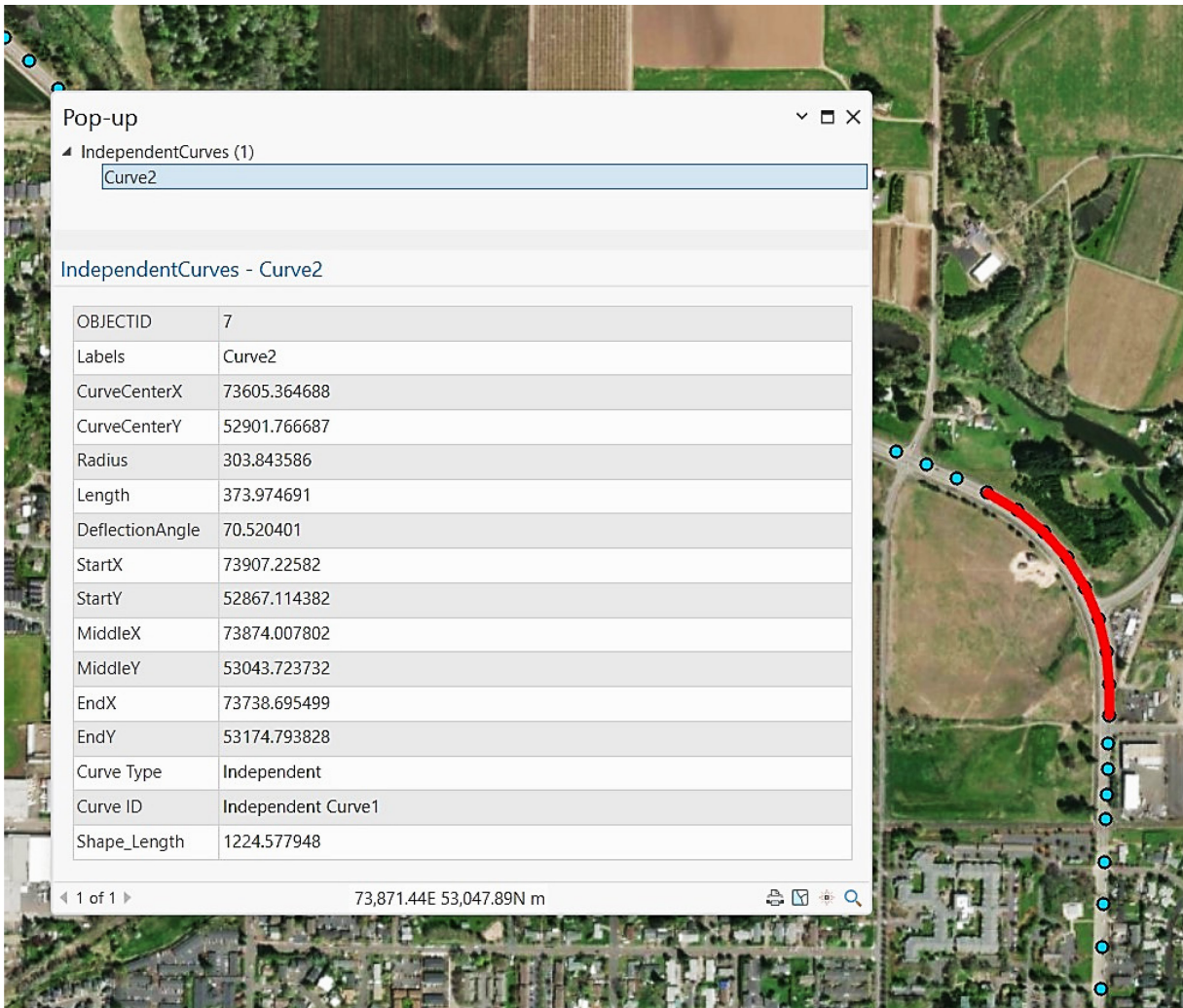


Figure B. 10 Curve characterization results of Horizontal toolbox for identified independent curve.

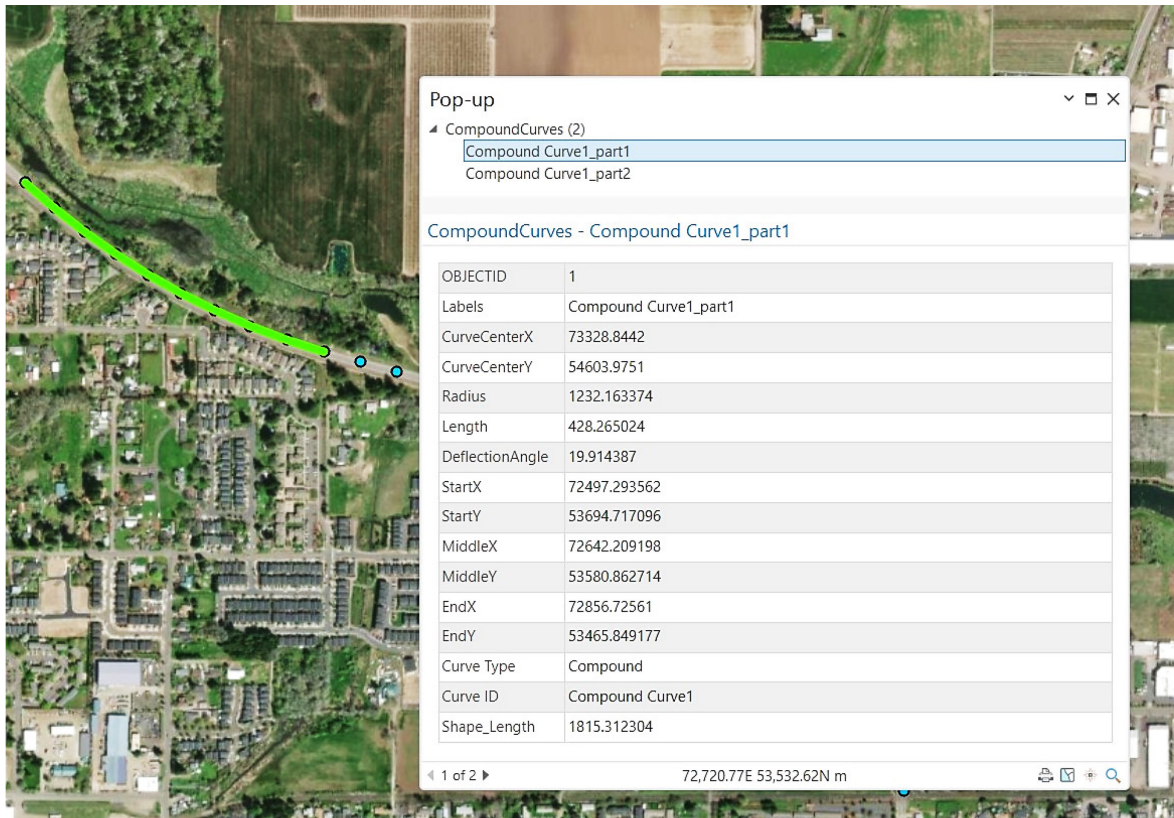


Figure B. 11 Curve characterization results of Horizontal toolbox for identified Compound curve.

Vertical Curve identification

Areas featuring vertical curves are displayed on the map using red polylines. Blue points represent down-sampled trajectory points that are visible on straight sections of the road. Also, an elevation profile (Z coordinate of trajectory points over index) of the trajectory can be displayed. This helps the user assess whether the selected parameters produce accurate results and align with the vertical curves. Each point in the map has an OBJECTID which matches with index in elevation profile. By examining the elevation profile and relating the Z coordinates to the corresponding OBJECTID or index, the user can verify whether a point with a specific OBJECTID identified as a vertical curve is indeed located on a vertical curve.

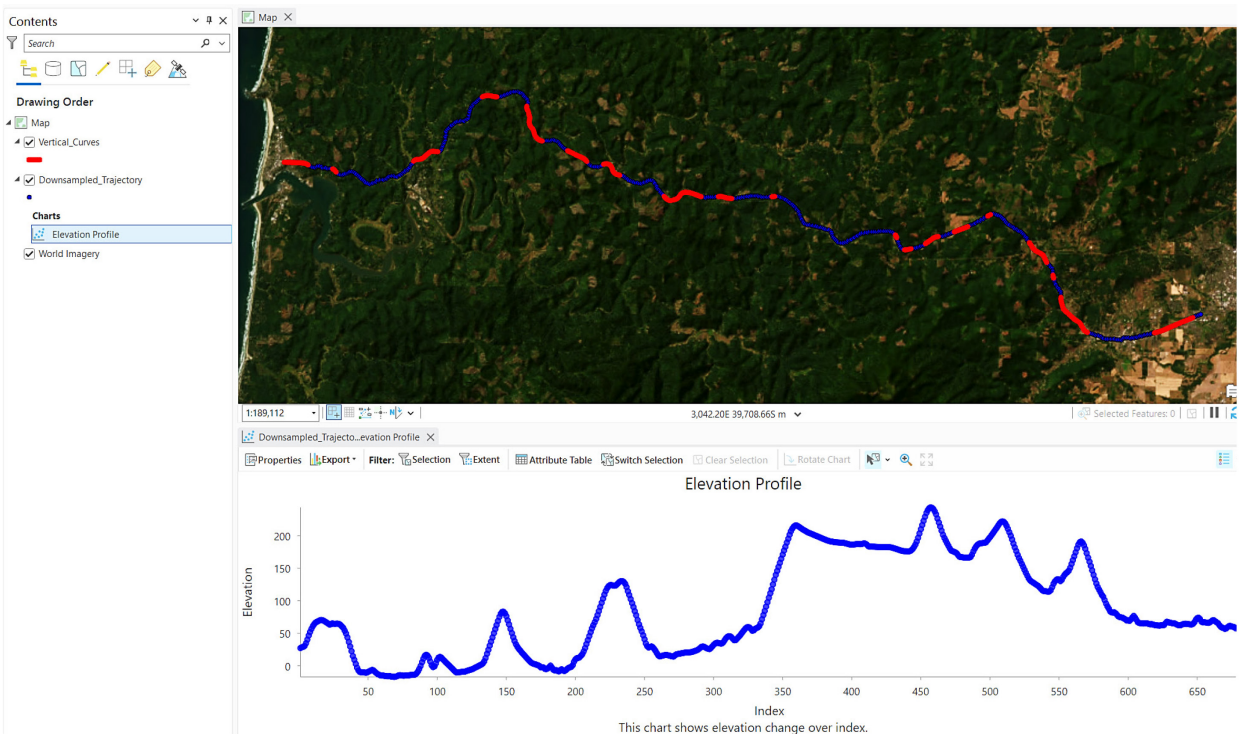


Figure B. 12 Curve identification results of Vertical toolbox for RIEGL_export_Mission dataset.

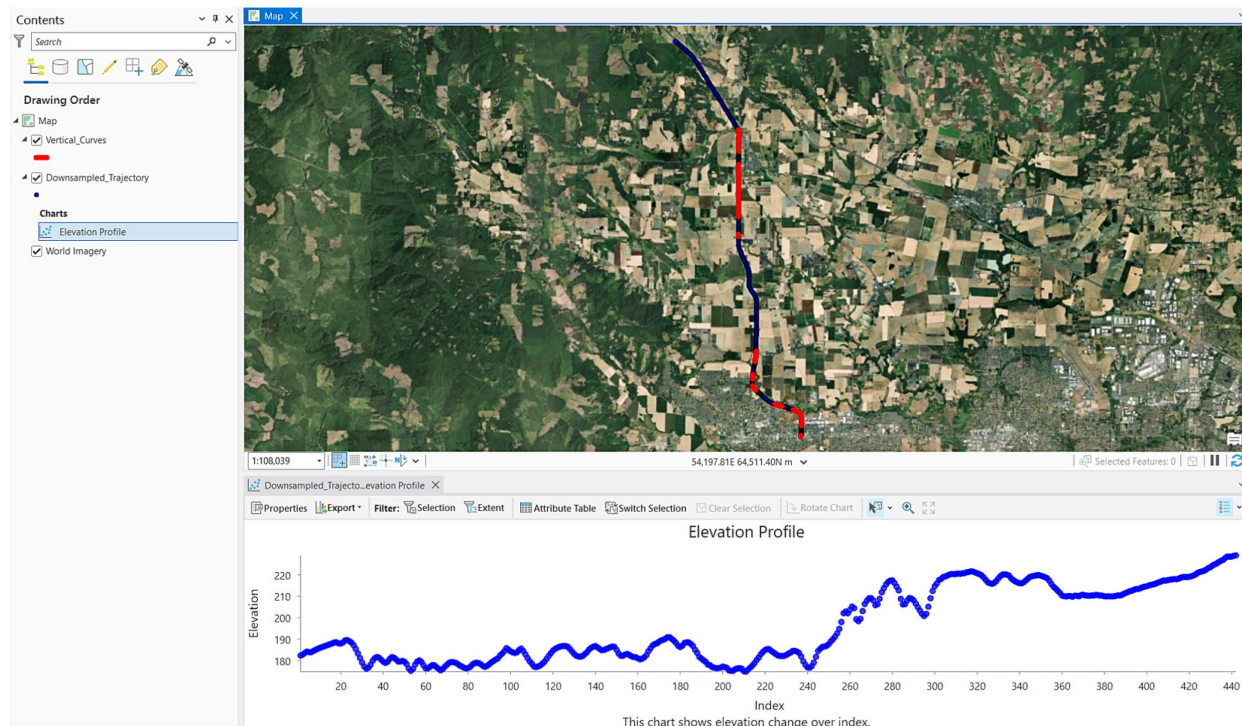


Figure B. 13 Curve identification results of Vertical toolbox for T01_HWY102_MP90-81 dataset.

Vertical Curve characterization

A pop-up window displaying the calculated characteristics of vertical curves will appear on the map when the user clicks on the curve polylines. These characteristics include coordinates of the curve start point, and end point (X and Y in meters) and curve length (in meter).

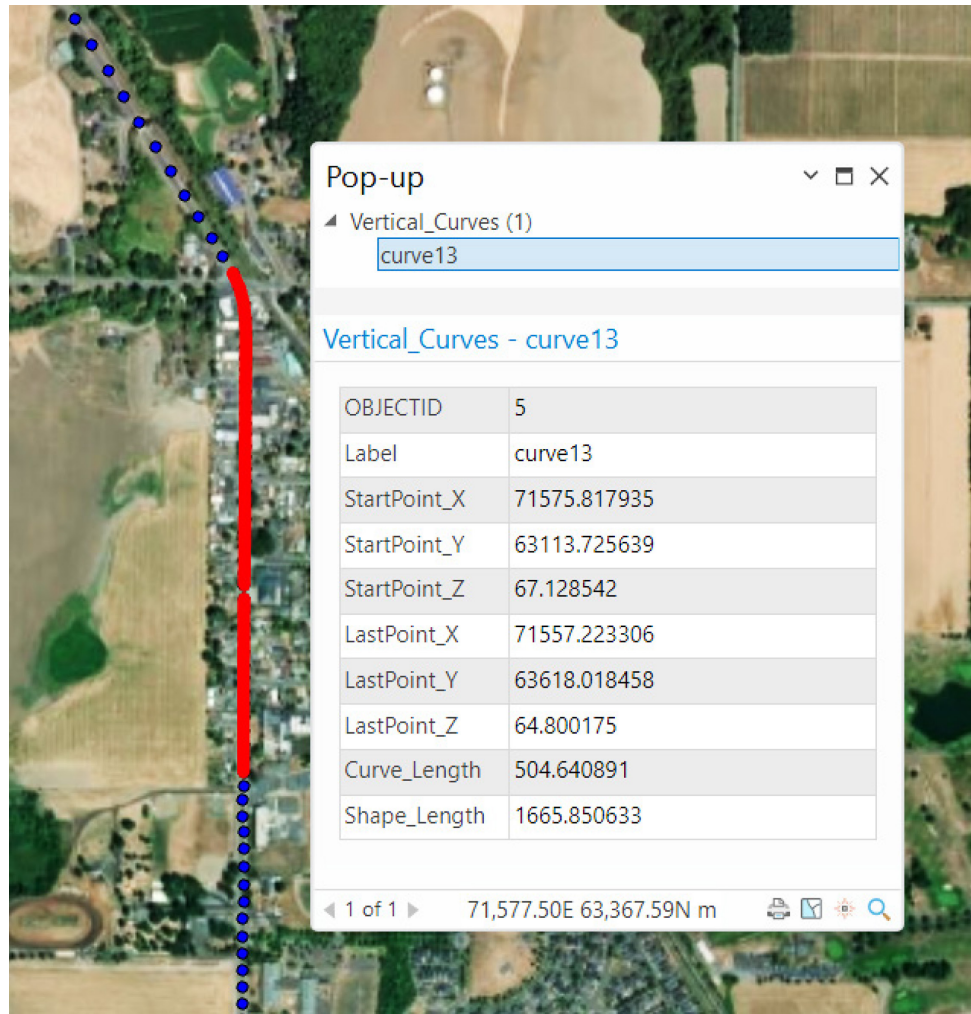


Figure B. 14 Curve characterization results of Vertical toolbox for identified curve.

Cross section extraction

Cross sections along the road profile (trajectory) are represented by blue polylines in the following figure. Each cross section has a unique ID such that the measurements and analysis results can be performed and recorded along with each individual cross section.

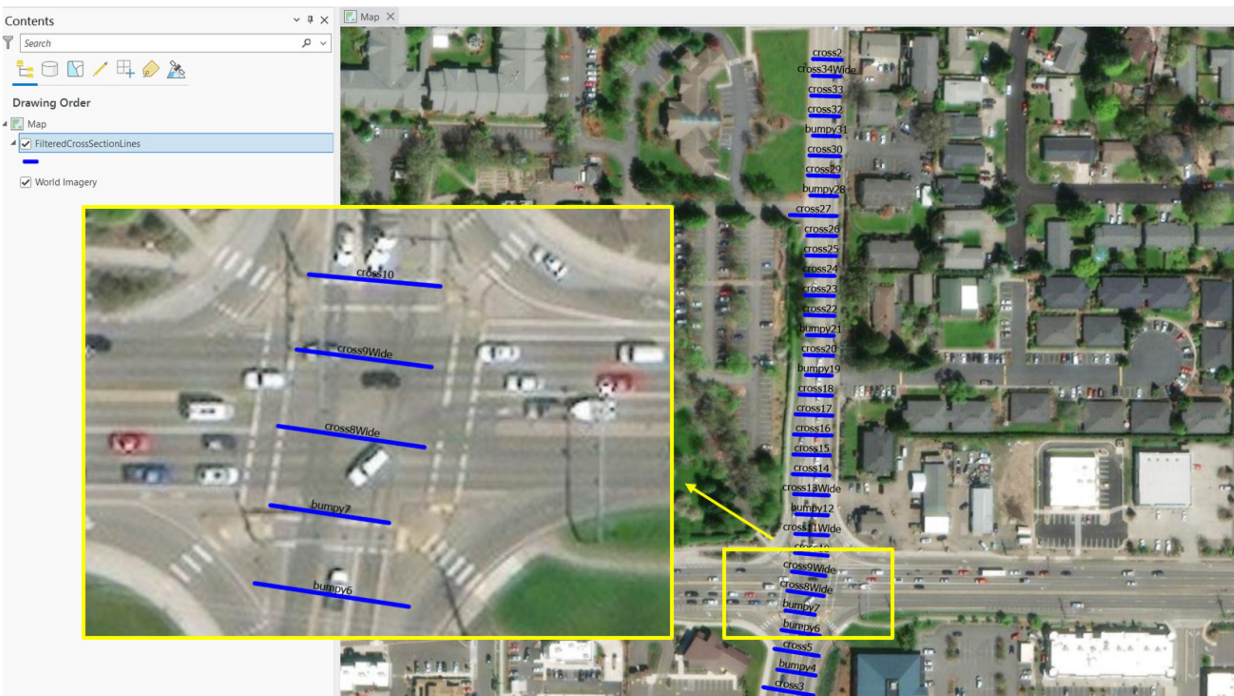


Figure B. 15 Cross Section extraction results of CrossSection toolbox for T01_HWY102_MP90-81 dataset.

Cross section characterization

By clicking on a cross section in ArcGIS Pro, a table will pop up displaying the cross-section characteristics. Cross sections located on a Straight ID have two cross slopes, extending from the crown point toward each side of the cross section. Cross sections located on a Curve ID have a single cross slope. The XYZ coordinates of the start and end points of each cross section are also provided in the table.

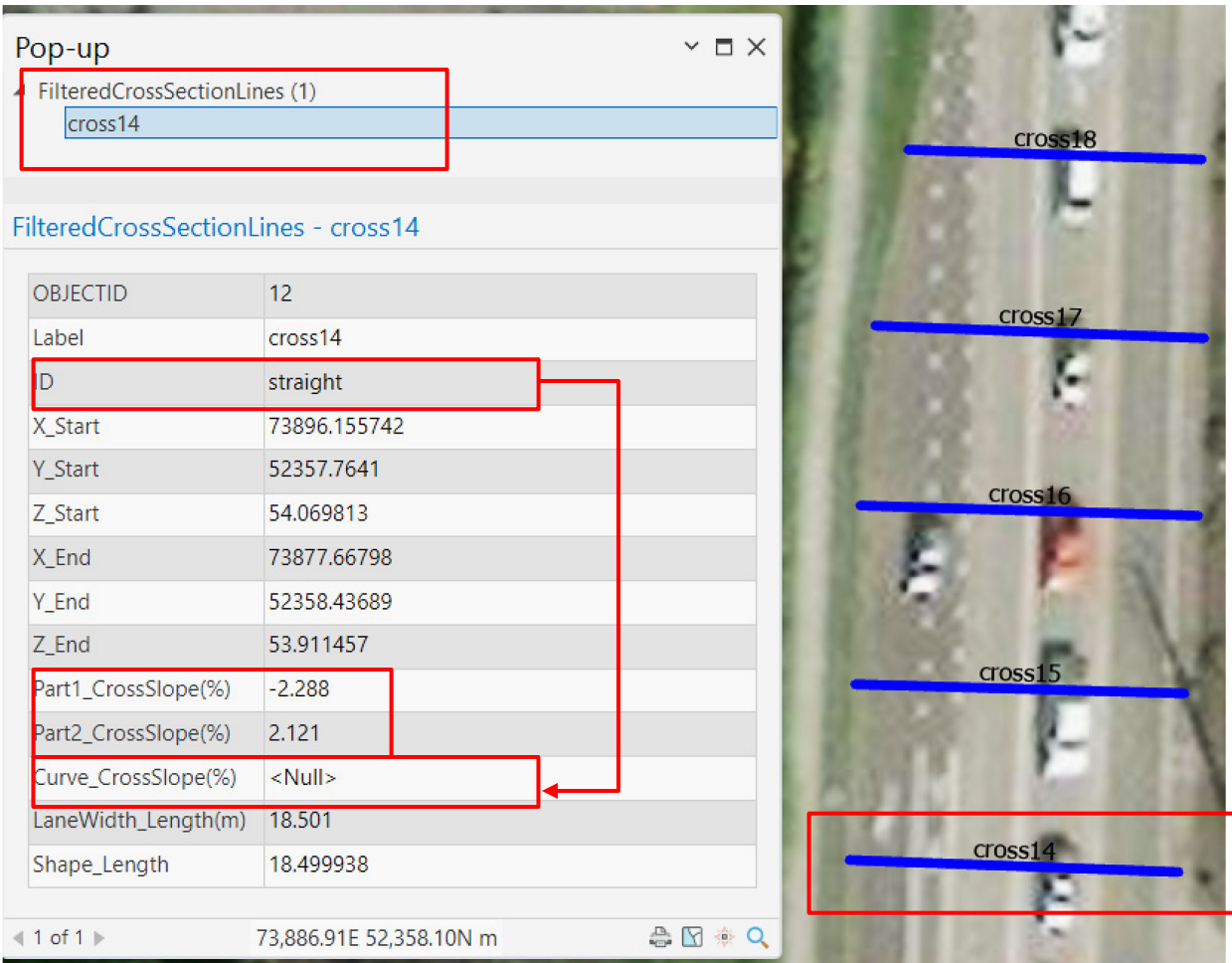


Figure B. 16 Cross Section characterization results of CrossSection toolbox for T01_HWY102_MP90-81 dataset.

KNOWN ISSUES

In the Horizontal Characterization toolbox, curves identified close to roundabouts, turns, and exits are not reliable. They may be identified and characterized incorrectly.

The Cross-Section Toolbox results may not be reliable when there are data gaps, or the asphalt is bumpy and cracked. Additionally, in areas where the detected road point cloud is incorrect or inaccurate, the results are unreliable.

- If one of the output or input files is open.
- If the required libraries are not installed.
- If input parameters are not within the typical range.
- If a gdal version error appears, comment the related line that checks the version from related script.

Here are some potential solutions when exceptions occur:

- Clear History
- Clean geodatabase by deleting previous results
- Restart ArcGIS Pro

FINE-TUNED PARAMETERS

The recommended input parameter values for each sample dataset in the Horizontal Characterization Tool are listed below.

Horizontal Curve

Sample Datasets by Parameter

Sample Datasets	Minimum Distance (ft)	GPS Time (seconds)	Lower Curvature Threshold	Upper Curvature Threshold	Coordinate System
RIEGL_export_Mission	30	2.5	0.5	15	NAD_1983_2011_UTM_Zone_10N
T01_HWY068_SB	100	2.5	0.15	3.5	OCRS_Portland_NAD_1983_2011_LCC_Feet_Intl
T01_HWY072_MP8-0	100	2.5	0.15	4.5	OCRS_Salem_NAD_1983_2011_TM_Feet_Intl
T01_HWY102_MP90-81	100	2.5	0.15	4.5	OCRS_Portland_NAD_1983_2011_LCC_Feet_Intl
T01_HWY150_MP20-15	100	2.5	0.1	3	OCRS_Salem_NAD_1983_2011_TM_Feet_Intl
T03_HWY029_MP20-0	100	2.5	0.13	3	OCRS_Portland_NAD_1983_2011_LCC_Feet_Intl
T03_HWY141	50	2	0.25	5	OCRS_Portland_NAD_1983_2011_LCC_Feet_Intl
T06_HWY091_MP0-20_SB	100	2.5	0.15	5	OCRS_Portland_NAD_1983_2011_LCC_Feet_Intl

Vertical Curve

The recommended input parameter values for each sample dataset in the Vertical Characterization Tool are listed below.

Sample Datasets by Parameter

Sample Datasets	Minimum Distance (ft)	GPS Time (seconds)	Curvature Threshold	Window Size	Coordinate System
RIEGL_export_Mission	100	2.5	0.3	5	NAD_1983_2011_UTM_Zone_10N
T01_HWY068_SB	30	1.5	0.3	4	OCRS_Portland_NAD_1983_2011_LCC Feet Intl
T01_HWY072_MP8-0	50	2	0.15	3	OCRS_Salem_NAD_1983_2011_TM Feet Intl
T01_HWY102_MP90-81	50	2	0.2	5	OCRS_Portland_NAD_1983_2011_LCC Feet Intl
T01_HWY150_MP20-15	50	2	0.2	5	OCRS_Salem_NAD_1983_2011_TM Feet Intl
T03_HWY029_MP20-0	100	2.5	0.3	5	OCRS_Portland_NAD_1983_2011_LCC Feet Intl
T03_HWY141	100	2.5	0.3	5	OCRS_Portland_NAD_1983_2011_LCC Feet Intl
T06_HWY091_MP0-20_SB	100	2.5	0.3	5	OCRS_Portland_NAD_1983_2011_LCC Feet Intl

APPENDIX C: LABEL RE-MAPPING GUI AND SOURCE

This appendix provides a brief overview of the python tool with its source script for efficient re-mapping of labeling schemes for various research needs. This tool was developed to change the classification labels of the training datasets to give additional flexibility in the training process as some of the objects can be assigned to multiple classes depending on the applications and focus. In this case, it was specifically designed and developed for the classification scheme proposed by the research team in this project (Table 3.3 and Table 3.4).

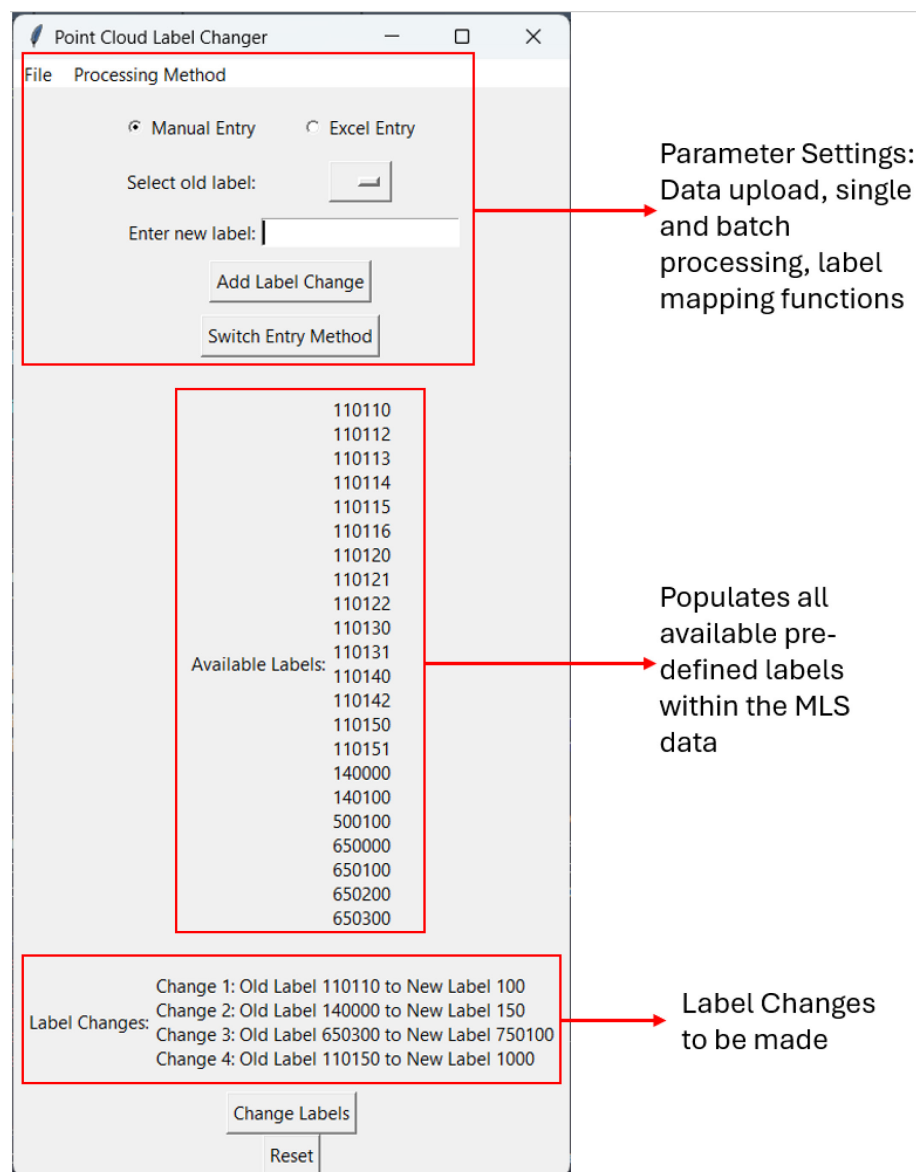


Figure C. 1 Screenshot of the GUI of the classification label change tool

Python Code for Classification Label Change Tool

```
# -*- coding: utf-8 -*-
"""
Created on Wed Dec 20 19:05:25 2023

@author: kenneyke
"""

import tkinter as tk
from tkinter import filedialog, messagebox
import laspy
import numpy as np
import pandas as pd
import os
import glob

class PointCloudLabelChanger:
    def __init__(self, root):
        self.processing_method = "Single" # Default to Single processing
method
        self.root = root
        self.root.title("Point Cloud Label Changer")

        # Create a menu bar
        menubar = tk.Menu(root)
        root.config(menu=menubar)

        # Create a file menu
        file_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="File", menu=file_menu)
        file_menu.add_command(label="Open Point Cloud",
command=self.open_point_cloud)
        file_menu.add_command(label="Load Excel Sheet",
command=self.load_excel_sheet) # Added for Excel loading

        # Create a processing method menu
        processing_method_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Processing Method",
menu=processing_method_menu)
        processing_method_menu.add_command(label="Single",
command=self.set_single_process)
        processing_method_menu.add_command(label="Batch",
command=self.set_batch_process)

        # Initialize variables
        self.point_cloud = None
        self.old_label_var = tk.StringVar()
        self.new_label_var = tk.StringVar()
        self.excel_file_path_var = tk.StringVar() # Add this line to
initialize the variable
        self.file_path = None

        # Create frame for process selection
        process_selection_frame = tk.Frame(root, padx=10, pady=10)
        process_selection_frame.pack()
```

```

# Create label selection frame
self.label_frame = tk.Frame(root, padx=10, pady=10)
self.label_frame.pack()

# Radio buttons for label change method
self.label_change_method_var = tk.StringVar()
self.label_change_method_var.set("Manual Entry") # Default to manual
entry
    manual_entry_radio = tk.Radiobutton(self.label_frame, text="Manual
Entry", variable=self.label_change_method_var, value="Manual Entry",
command=self.update_label_entry_method)
    manual_entry_radio.grid(row=0, column=0, pady=5)

    excel_entry_radio = tk.Radiobutton(self.label_frame, text="Excel
Entry", variable=self.label_change_method_var, value="Excel Entry",
command=self.update_label_entry_method)
    excel_entry_radio.grid(row=0, column=1, pady=5)

    old_label_frame_label = tk.Label(self.label_frame, text="Select old
label:")
    old_label_frame_label.grid(row=1, column=0, pady=5)

    # Dropdown button for selecting old label
    self.old_label_dropdown = tk.OptionMenu(self.label_frame,
self.old_label_var, *[""])
    self.old_label_dropdown.grid(row=1, column=1, pady=5)
    self.old_label_var.set("")

    new_label_frame_label = tk.Label(self.label_frame, text="Enter new
label:")
    new_label_frame_label.grid(row=2, column=0, pady=5)

    self.new_label_entry = tk.Entry(self.label_frame,
textvariable=self.new_label_var)
    self.new_label_entry.grid(row=2, column=1, pady=5)

# Create button to add label change
add_label_change_button = tk.Button(self.label_frame, text="Add Label
Change", command=self.add_label_change)
add_label_change_button.grid(row=3, column=0, columnspan=2, pady=5)

# Create label display frame
label_display_frame = tk.Frame(root, padx=10, pady=10)
label_display_frame.pack()

label_display_frame_label = tk.Label(label_display_frame,
text="Available Labels:")
label_display_frame_label.grid(row=0, column=0, pady=5)

self.available_labels_var = tk.StringVar()
label_display_frame_data = tk.Label(label_display_frame,
textvariable=self.available_labels_var, justify=tk.LEFT)
label_display_frame_data.grid(row=0, column=1, pady=5)

# Create frame to display label changes
label_changes_frame = tk.Frame(root, padx=10, pady=10)

```

```

label_changes_frame.pack()

label_changes_frame_label = tk.Label(label_changes_frame, text="Label
Changes:")
label_changes_frame_label.grid(row=0, column=0, pady=5)

self.label_changes_var = tk.StringVar()
label_changes_frame_data = tk.Label(label_changes_frame,
textvariable=self.label_changes_var, justify=tk.LEFT)
label_changes_frame_data.grid(row=0, column=1, pady=5)

# Button to change labels
change_labels_button = tk.Button(root, text="Change Labels",
command=self.change_label)
change_labels_button.pack()

# Create a button to switch between Manual and Excel Entry methods
switch_method_button = tk.Button(self.label_frame, text="Switch Entry
Method", command=self.switch_label_entry_method)
switch_method_button.grid(row=4, column=0, columnspan=2, pady=5)

# Create a button to reset the application
reset_button = tk.Button(root, text="Reset", command=self.reset)
reset_button.pack()

# Initialize a list to store label change inputs
self.label_changes = []

# Add a method to update the selected processing method
def update_processing_method(self, method):
    self.processing_method = method
    messagebox.showinfo("Processing Method", f"Processing method set to:
{method}")

def set_single_process(self):
    self.update_processing_method("Single")
    # Add your logic for the "Single" processing method here
    print("Single processing method selected")

def set_batch_process(self):
    self.update_processing_method("Batch")
    # Add your logic for the "Batch" processing method here
    print("Batch processing method selected")

def reset(self):
    # Reset all variables and clear GUI components
    self.point_cloud = None
    self.old_label_var.set("")
    self.new_label_var.set("")
    self.excel_file_path_var.set("")
    self.file_path = None
    self.label_changes = []

    # Reset the label selection frame
    self.old_label_var.set("")
    self.new_label_var.set("")
    self.old_label_dropdown.config(state="normal")

```



```

self.new_label_entry.config(state="normal")

# Reset the label display frame
self.available_labels_var.set("")

# Reset the label changes frame
self.label_changes_var.set("")

# Update the dropdown menu with available labels
labels = self.get_unique_labels()
menu = self.old_label_dropdown["menu"]
menu.delete(0, "end")
for label in labels:
    menu.add_command(label=label, command=lambda l=label:
self.old_label_var.set(l))

def switch_label_entry_method(self):
    # Method to switch between Manual and Excel Entry methods
    current_method = self.label_change_method_var.get()
    if current_method == "Manual Entry":
        self.label_change_method_var.set("Excel Entry")
        self.update_label_entry_method()
    elif current_method == "Excel Entry":
        # Reload the original labels when switching back to Manual Entry
        self.label_change_method_var.set("Manual Entry")
        self.update_label_entry_method()
        labels = self.get_unique_labels()
        # Update the dropdown menu with available labels
        menu = self.old_label_dropdown["menu"]
        menu.delete(0, "end")
        for label in labels:
            menu.add_command(label=label, command=lambda l=label:
self.old_label_var.set(l))

def update_label_entry_method(self):
    method = self.label_change_method_var.get()
    if method == "Manual Entry":
        self.old_label_dropdown.config(state="normal")
        self.old_label_var.set("")
        self.new_label_entry.config(state="normal")
        self.new_label_var.set("")

    elif method == "Excel Entry":
        self.old_label_dropdown.config(state="disabled")
        self.old_label_var.set("")
        self.new_label_entry.config(state="disabled")
        self.new_label_var.set("")
        # Prompt the user about the Excel format
        messagebox.showinfo("Excel Format", "Please ensure that the Excel
sheet has *old labels* in the first column and *new labels* in the second
column with their associated headings.")

def load_excel_sheet(self):
    file_path = filedialog.askopenfilename(filetypes=[("Excel Files",
"*.xlsx;*.xls")])

```

```

        if file_path:
            self.excel_file_path_var.set(file_path)
            messagebox.showinfo("Excel Sheet Loaded", f"Excel sheet
loaded:\n{file_path}")

            # Optionally, you can read the Excel file here and display the
            labels in the GUI
            try:
                excel_data = pd.read_excel(file_path)
                old_labels = excel_data.iloc[:, 0].astype(int)
                new_labels = excel_data.iloc[:, 1].astype(int)

                # Display label changes in the GUI
                changes_str = "\n".join([f"Change {i+1}: Old Label {old} to
New Label {new}"
                                         for i, (old, new) in
enumerate(zip(old_labels, new_labels))])
                self.label_changes_var.set(changes_str)

                # Append the new label changes to the existing list
                self.label_changes.extend(list(zip(old_labels, new_labels)))

                # Update the dropdown menu with available old labels
                menu = self.old_label_dropdown["menu"]
                menu.delete(0, "end")
                for label in old_labels:
                    menu.add_command(label=label, command=lambda l=label:
self.old_label_var.set(l))

            except Exception as e:
                messagebox.showerror("Error", f"Error reading Excel file:
{e}")

    def open_point_cloud(self):
        file_path = filedialog.askopenfilename(filetypes=[("Point Cloud
Files", "*.las;*.laz")])
        if file_path:
            self.file_path = file_path
            self.point_cloud = self.read_point_cloud(file_path)

            # Check the selected processing method
            if self.processing_method == "Single":
                # Show available labels for the single selected file
                labels = self.get_unique_labels_single_file()
            elif self.processing_method == "Batch":
                # Show the cumulative available labels for all files
                labels = self.get_unique_labels_cumulative()
            print(labels)
            label_str = '\n'.join(map(str, labels))
            self.available_labels_var.set(label_str)

            # Update the dropdown menu with available labels
            menu = self.old_label_dropdown["menu"]
            menu.delete(0, "end")
            for label in labels:

```

```

        menu.add_command(label=label, command=lambda l=label:
self.old_label_var.set(l))

        messagebox.showinfo("Available Labels", f"Available
labels:\n{label_str}")

    def get_unique_labels_single_file(self):
        if self.point_cloud is not None:
            return np.unique(self.point_cloud[:, -1]).astype(int)
        else:
            return []

    def get_unique_labels_cumulative(self):
        # Get unique labels across all loaded point clouds
        unique_labels_set = set()
        for original_ptcloud_path in self.get_loaded_files():
            original_ptcloud = laspy.read(original_ptcloud_path)
            current_labels =
np.unique(original_ptcloud.Ext_Class).astype(int)
            unique_labels_set.update(current_labels)
        return sorted(list(unique_labels_set))

    def get_loaded_files(self):
        # Returns the list of loaded point cloud files (used for batch
processing)
        return [self.file_path] if self.processing_method == "Single"
else glob.glob(os.path.join(os.path.dirname(self.file_path), '*.laz'))

    def add_label_change(self):
        method = self.label_change_method_var.get()
        if method == "Manual Entry":
            # Manual entry logic
            old_label = self.old_label_var.get()
            new_label = self.new_label_var.get()
        elif method == "Excel Entry":
            # Excel entry logic
            file_path = self.excel_file_path_var.get()
            if not file_path:
                messagebox.showerror("Error", "Please provide an Excel
file.")
            return

        # Read the Excel file and extract label changes
        try:
            excel_data = pd.read_excel(file_path)
            old_labels = excel_data.iloc[:, 0].astype(int)
            new_labels = excel_data.iloc[:, 1].astype(int)

            # Display label changes in the GUI
            changes_str = "\n".join([f"Change {i+1}: Old Label {old} to
New Label {new}"]
                                for i, (old, new) in
enumerate(zip(old_labels, new_labels))])
            self.label_changes_var.set(changes_str)

            # Update the label_changes list
            self.label_changes = list(zip(old_labels, new_labels))

```

```

        except Exception as e:
            messagebox.showerror("Error", f"Error reading Excel file:
{e}")

        return

    # Add new label change mapping to the list
    old_label = self.old_label_var.get()
    new_label = self.new_label_var.get()

    if old_label and new_label:
        self.label_changes.append((old_label, new_label))

    # Update the label_changes_var to display the entered mappings
    changes_str = "\n".join([f"Change {i + 1}: Old Label {old} to New
Label {new}"
                                for i, (old, new) in
enumerate(self.label_changes)])
    self.label_changes_var.set(changes_str)

    # Clear the entry fields for the next mapping
    self.old_label_var.set("")
    self.new_label_var.set("")

def read_point_cloud(self, file_path):
    return self.readPtcloud(file_path)

def readPtcloud(self, file_path):
    L = laspy.read(file_path)
    # Extracting X, Y, Z, Intensity, and Ext_Class
    ptcloud = np.array((L.x, L.y, L.z, L.intensity,
L.Ext_Class)).transpose()
    return ptcloud

def read_Original_Ptcloud(self, file_path):
    LL = laspy.read(file_path)
    originalptcloud = np.array((LL.x, LL.y, LL.z, LL.intensity,
LL.return_number, LL.number_of_returns, LL.scan_direction_flag,
LL.edge_of_flight_line, LL.classification, LL.user_data, LL.point_source_id,
LL.gps_time, LL.red, LL.green, LL.blue)).transpose()
    return originalptcloud

def get_unique_labels(self):
    if self.point_cloud is not None:
        return np.unique(self.point_cloud[:, -1]).astype(int)
    else:
        return []

def change_label(self):
    if self.point_cloud is not None:
        try:
            # Check if label changes are provided from Excel
            if self.label_changes:
                #Also Checking the Processing Method
                if self.processing_method == "Single":
                    self.process_single_method()
                elif self.processing_method == "Batch":
                    self.process_batch_method()

```

```

        messagebox.showinfo("Label Change",
f"{len(self.label_changes)} label changes applied to all files. Point cloud
saved.")

        return # Stop further execution after processing
batch method

    else:
        print("Processing method not selected")

        # Iterate through the list of label changes from Excel
        for i, (old_label, new_label) in
enumerate(self.label_changes, start=1):
            # Convert labels to int
            try:
                old_label = int(old_label)
                new_label = int(new_label)
            except ValueError:
                messagebox.showerror("Error", "Invalid label
format. Please enter valid integers.")
                return

            # Check if the old label exists
            if old_label not in self.get_unique_labels():
                messagebox.showerror("Error", f"Selected label
{old_label} does not exist.")
                return

            # Modify the Ext_Class field
            mask = self.point_cloud[:, -1] == old_label
            self.point_cloud[mask, -1] = new_label

        elif self.label_changes:
            # Iterate through the list of manually entered label
changes
            for i, (old_label, new_label) in
enumerate(self.label_changes, start=1):
                # Convert labels to int
                try:
                    old_label = int(old_label)
                    new_label = int(new_label)
                except ValueError:
                    messagebox.showerror("Error", "Invalid label
format. Please enter valid integers.")
                    return

                # Check if the old label exists
                if old_label not in self.get_unique_labels():
                    messagebox.showerror("Error", f"Selected label
{old_label} does not exist.")
                    return

                # Modify the Ext_Class field
                mask = self.point_cloud[:, -1] == old_label
                self.point_cloud[mask, -1] = new_label

            # Save the updated point cloud
            file_path, _ = os.path.splitext(self.file_path) # Get the
file path without extension

```

```

        save_path =
filedialog.asksaveasfilename(defaultextension=".laz",
                                filetypes=[("LAS
Files", "*.laz")],

initialfile=f"{file_path}_updated.laz")
        if save_path:
            self.write_las(save_path, self.point_cloud)
            messagebox.showinfo("Label Change",
f"{len(self.label_changes)} label changes applied. Point cloud saved.")
        else:
            messagebox.showwarning("Warning", "Save operation
canceled. Point cloud reverted.")
        except ValueError:
            messagebox.showerror("Error", "Invalid input. Please enter
valid labels.")

def write_las(self, file_path, point_cloud):
    # Read the original LAS file
    original_ptcloud_path = self.file_path
    original_ptcloud = laspy.read(original_ptcloud_path)

    # Iterate through the list of label changes
    for old_label, new_label in self.label_changes:
        print(f"Attempting to change label: {old_label} to {new_label}")

        try:
            old_label = int(old_label)
            new_label = int(new_label)
        except ValueError:
            print("Invalid label format.")
            messagebox.showerror("Error", "Invalid label format. Please
enter valid labels.")
            return

        print(f"Converted labels to int: {old_label} to {new_label}")

        # Check if the old label exists
        available_labels = self.get_unique_labels()
        print(f"Available labels: {available_labels}")

        # Modify the Ext_Class field
        mask = np.isclose(original_ptcloud.Ext_Class, old_label)
        original_ptcloud.Ext_Class[mask] = new_label

    # Save the updated point cloud with new labels
    file_path, _ = os.path.splitext(self.file_path)
    updated_file_path = f"{file_path}_updated.laz"
    original_ptcloud.write(updated_file_path)

    # Writing out the new point cloud
    out_las = laspy.create(file_version="1.4", point_format=7)
    out_las.header.offset = [original_ptcloud.x.min(),
original_ptcloud.y.min(), original_ptcloud.z.min()]
    out_las.header.scale = [0.1, 0.1, 0.1]
    out_las.x = original_ptcloud.x
    out_las.y = original_ptcloud.y

```

```

out_las.z = original_ptcloud.z
out_las.intensity = original_ptcloud.intensity
out_las.red = original_ptcloud.red
out_las.green = original_ptcloud.green
out_las.blue = original_ptcloud.blue
out_las.gps_time = original_ptcloud.gps_time
out_las.classification = original_ptcloud.classification

# Ensure that 'Ext_Class' attribute is available in 'point_cloud'
if 'Ext_Class' in original_ptcloud.point_format.dimension_names:
    ext_class_index =
list(original_ptcloud.point_format.dimension_names).index('Ext_Class')
    if ext_class_index < point_cloud.shape[1]:
        out_las.Ext_Class = point_cloud[:, ext_class_index]
%% Function for Single / Batch Process
#Single Processing
def process_single_method(self):
    print("Processing Single method")
    # Logic for the "Single" processing method

#####Batch Processing #####
def process_batch_method(self):
    print("Processing Batch method")
    # Logic for the "Batch" processing method
    directory = os.path.dirname(self.file_path)
    file_pattern = os.path.join(directory, '*.laz')
    files = glob.glob(file_pattern)

    # # Initialize a set to store unique labels across all files
    # unique_labels_set = set()

    for original_ptcloud_path in files:
        #Read each point cloud file
        original_ptcloud = laspy.read(original_ptcloud_path)

        # Apply label changes to each point cloud
        self.batch_label_change_save(original_ptcloud,
original_ptcloud_path)

    def batch_label_change_save(self, original_ptcloud, file_path):
        # Iterate through the list of label changes
        for old_label, new_label in self.label_changes:
            print(f"Attempting to change label: {old_label} to {new_label}")

            try:
                old_label = int(old_label)
                new_label = int(new_label)
            except ValueError:
                print("Invalid label format.")
                messagebox.showerror("Error", "Invalid label format. Please
enter valid labels.")
                return

            print(f"Converted labels to int: {old_label} to {new_label}")

            # Check if the old label exists
            available_labels = self.get_unique_labels()

```

```

    print(f"Available labels: {available_labels}")

    # Modify the Ext_Class field
    mask = np.isclose(original_ptcloud.Ext_Class, old_label)
    original_ptcloud.Ext_Class[mask] = new_label

    # Save the updated point cloud with new labels
    file_path, _ = os.path.splitext(file_path)
    updated_file_path = f"{file_path}_updated.laz"

    original_ptcloud.write(updated_file_path)

if __name__ == "__main__":
    root = tk.Tk()
    app = PointCloudLabelChanger(root)
    root.mainloop()

```


APPENDIX D: ADDITIONAL SUMMARY STATISTICS OF ACCURACY ASSESSMENT FOR MACHINE LEARNING-BASED CLASSIFICATION

This appendix provides additional statistical summary of the accuracy assessment conducted by the research team in the study investigating the performance of machine learning approaches in classification and the impacts of the training datasets.

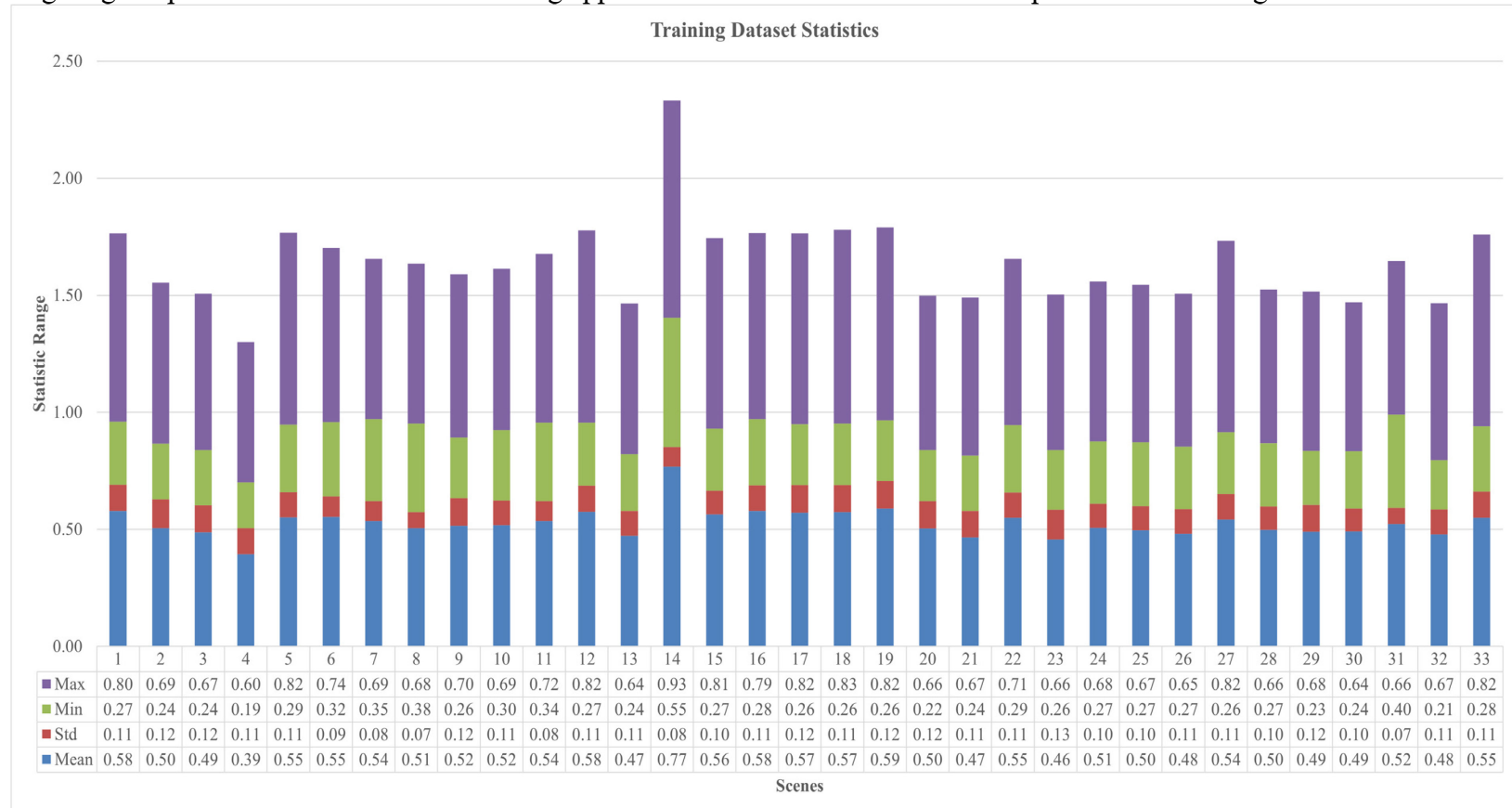


Figure D. 1 Random Forrest summary statistics of external accuracies (performances) for each scene for all 33 scenes as training datasets

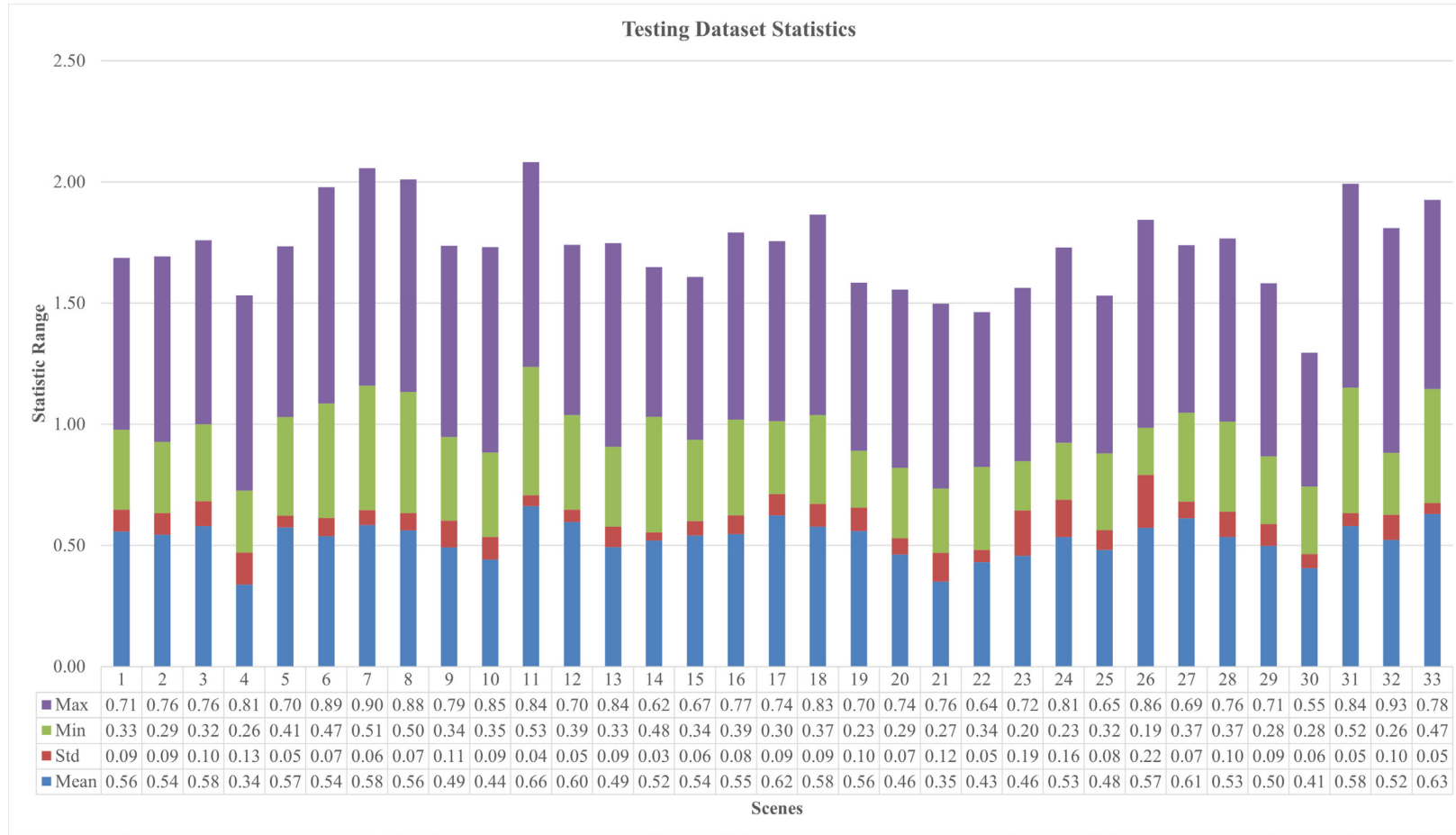


Figure D. 2 Random Forrest summary statistics of external accuracies (performances) for each scene for all 33 scenes as testing datasets

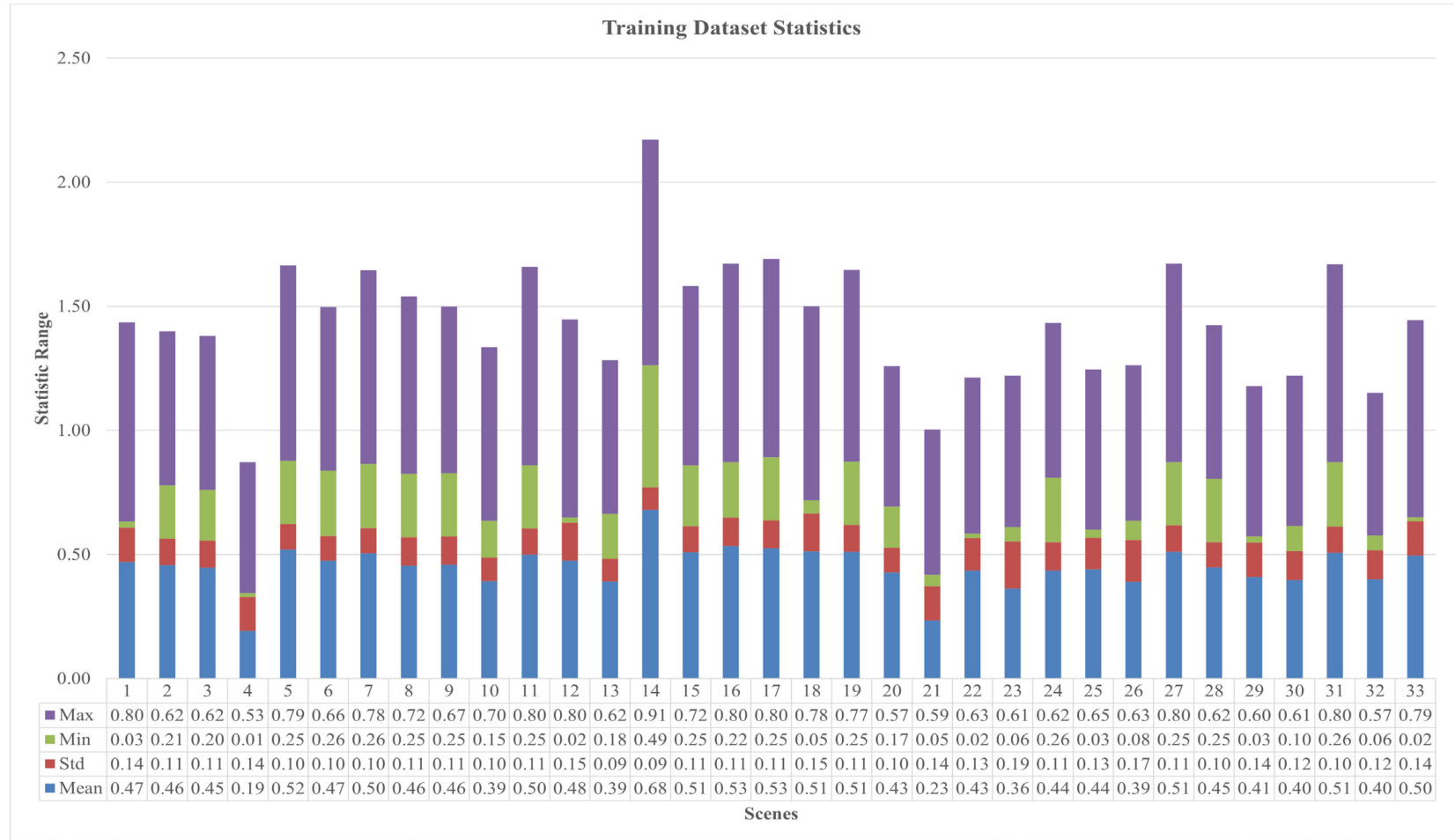


Figure D. 3 Neural Network summary statistics of external accuracies (performances) for each scene for all 33 scenes as training datasets

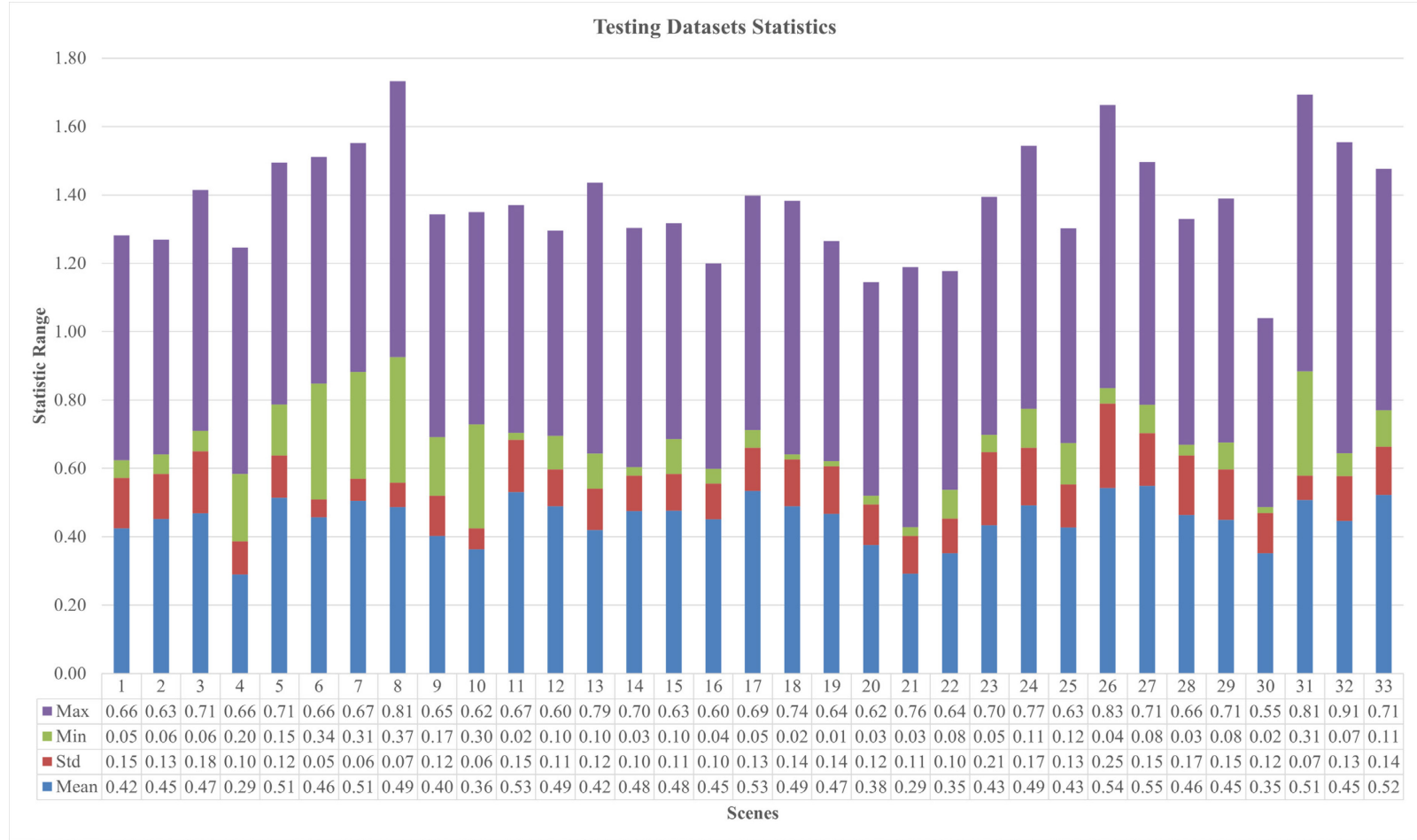


Figure D. 4 Neural Network summary statistics of external accuracies (performances) for each scene for all 33 scenes as testing datasets



Figure D. 5 Random Forrest summary statistics of external accuracies (performances) for each site for all 5 sites as training datasets



Figure D. 6 Random Forrest summary statistics of external accuracies (performances) for each site for all 5 sites as testing datasets



Figure D. 7 Neural Network summary statistics of external accuracies (performances) for each site for all 5 sites as training datasets

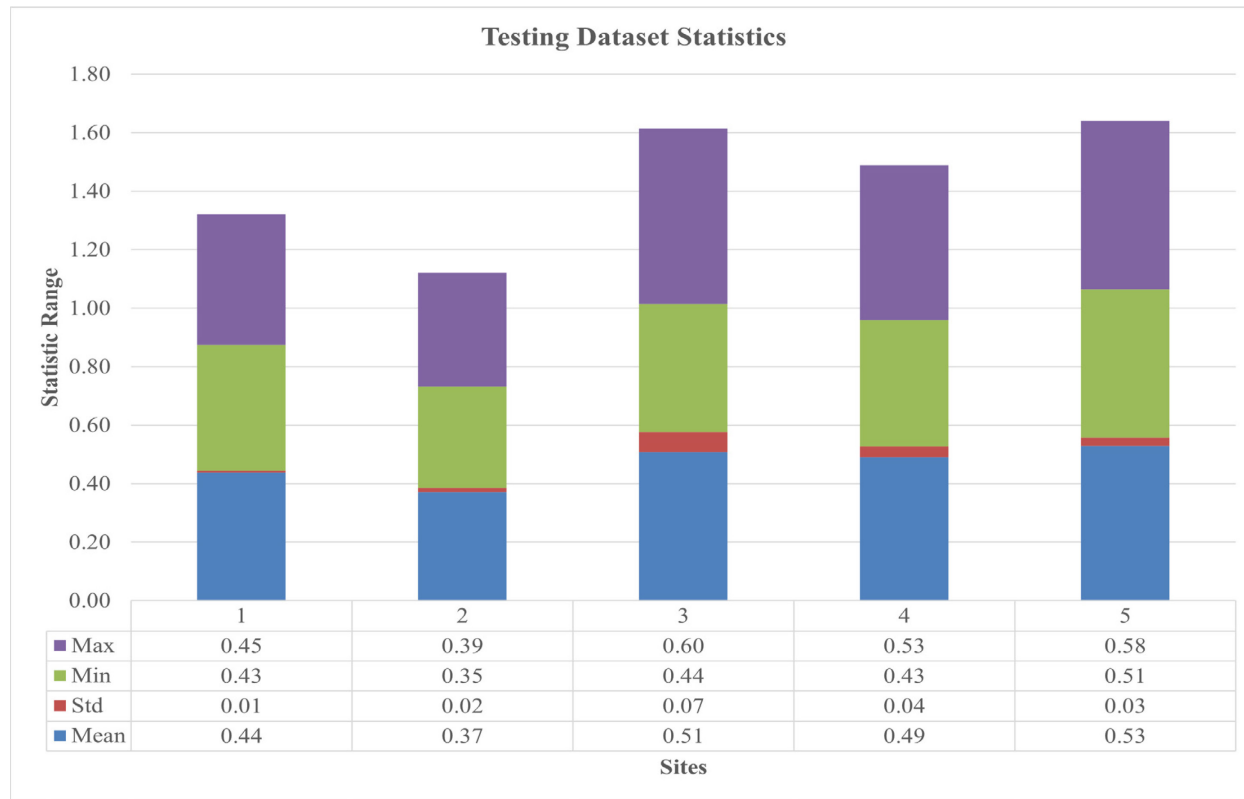


Figure D. 8 Neural Network summary statistics of external accuracies (performances) for each site for all 5 sites as testing datasets

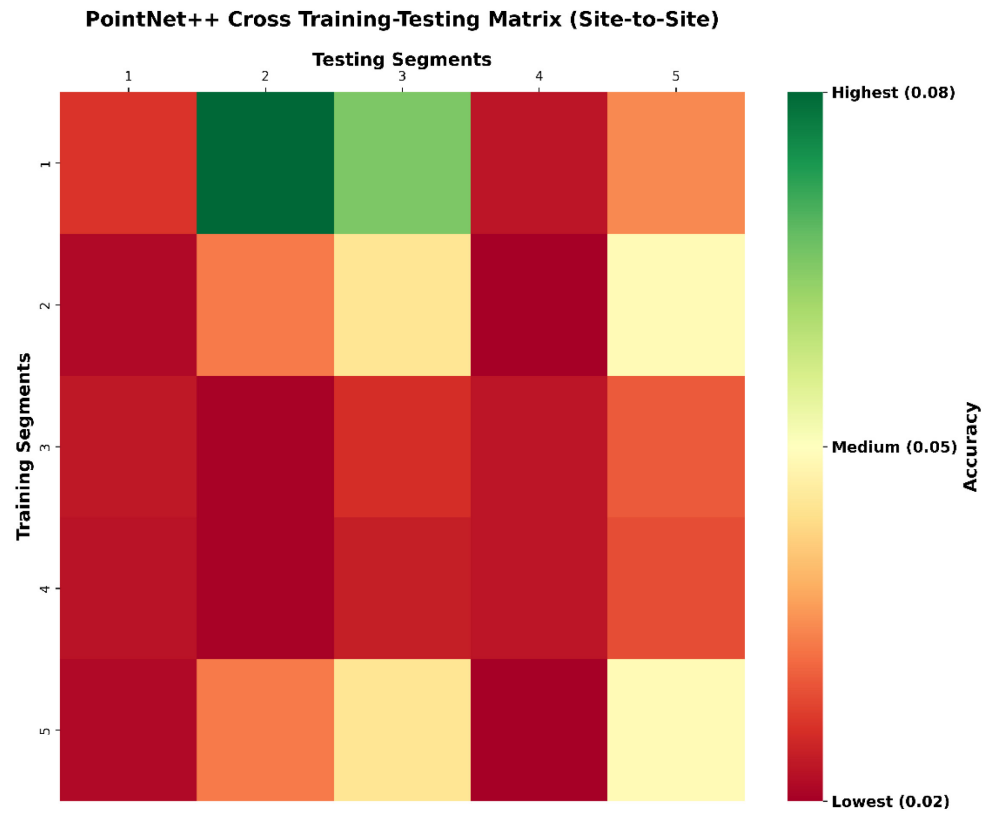


Figure D. 9 Site-to-site cross training and testing result for PointNet++ which failed under this workflow