GPR Data Visualizer Manual

Development Team: Benjamin Day, Skylaar Mease, R. Clay Shriver

Dev Team Supervisor: Dr. James Bryce

Developed with Funding from the US Army Corp of Engineers

Last Edited: 4/30/2021

Table of Contents

Data Transform Implementation

Output

Future Development

Index of Images

Installation

System Requirements

This application requires that the user's machine has an installation of Anaconda. Anaconda is a Python environment that assists in managing the necessary packages and dependencies for the software. Anaconda is further explained on page Page 12, and can be found at the following link: https://www.anaconda.com/products/individual

The application will come in a compressed file format. This folder can be unzipped using tools available built into the operating system or by using an entirely separate tool such as WinRAR or 7Zip. Regardless of choice, the user needs to be able to unzip a file.

Finally, the application must be installed on a machine running the Windows operating system.

Where to get the Application

There are two main ways to get the application onto your machine: have it shared to you by someone, or download it from the online repository.

When a coworker sends you the program over email, you will find a compressed folder in the attachments. You should be able to download the folder from that email in some way. Some email services may require you to follow a link and download from an online storage site such as Google Drive or OneDrive. Others will allow you to download the folder directly from the email itself.

When you want to download the application from the repository, you will need to first navigate to this link: https://github.com/GPR-Data-Visualizer/GPR-Data-Visualizer

Once you are on the repository page, you should see a green button with a download symbol and the word "code" on it. Clicking this button will open a dropdown menu. Select the option "Download ZIP" and your download will begin.

How to Install the Environment

Installing the environment for the application to run in is all done inside of the folder you downloaded in the previous step. The first step is to decompress it.

Open a file browser and navigate to where the application was downloaded to, more than likely you will find it in the default Downloads folder on your machine. Once found, right click the ZIP folder and select the Extract All option. Select a destination location on the next window, and your unzipped files will be placed there. If you are using an external software such as WinRAR, please follow that application's procedures to unzip your files.

Once the file is unzipped, you will need to run the installation code. Navigate to the now unzipped folder and look for the Install_GPR_Visualizer.bat file inside. Double click that file. You should see an instance of the command prompt open. This is the installer working to download all required packages onto the machine. The process may take some time, so please wait until the installation is complete. As a result, there should be a new script in the folder, and a new icon on your desktop. Once the installation process is over, the application should open automatically for the first time. If it does, then the installation was a success.

Operation

Running the Software

The installation process will have created a shortcut icon on your desktop. Double click this icon to start the application. You can also find the location of the installed folder on your machine, and run the new script created by the installer.

Software on Open

The software will boot into what is referred to as the Home Tab. This tab will always be opened on startup, and simply contains a few reminders for users of the application. In the future this tab may contain update information should a new release be shared, or may be used to share more reminders to users. The Home Tab cannot be closed and will remain open as long as the software is running.

Opening a New Tab

Opening a new tab in the application is done using the plus icon. Look towards the top of the application window for the tab labelled Home; the plus icon will initially be next to that original tab. As you open or close tabs, the plus icon will be shifted to be the rightmost tab at all times.

When you click the plus icon, the application will open up the file explorer native to your operating system. You will be prompted to select files to be opened and read for data

visualization. You can select a single file just by clicking a file. You can also select multiple files by either clicking the first file in the list and clicking the last file while holding the shift key or by clicking and dragging the mouse to area-select the files. Once the files are selected, click open and a new tab will be open in the application with the selected data. You can cancel the file selection process by clicking the cancel button or closing the file select window with the X icon.

Breakdown of a Visualizer Tab

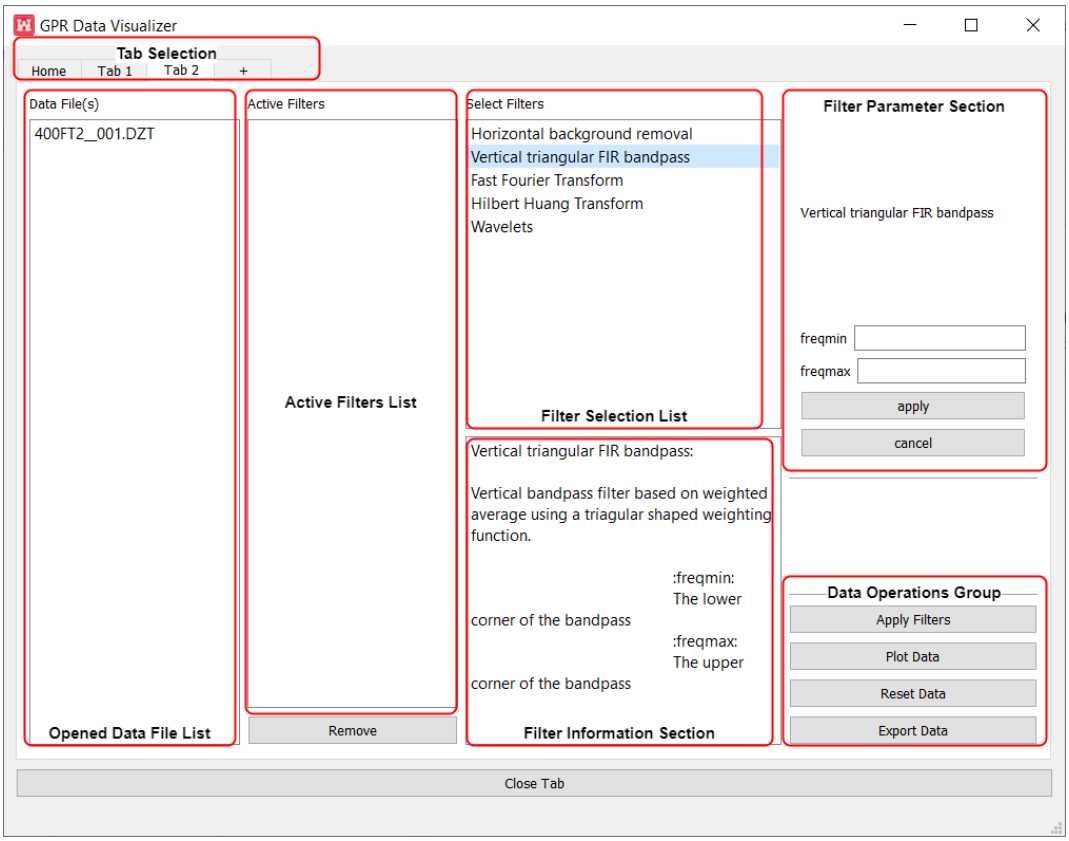

Figure 1: A screencap of the application with different operational sections outlined

Visualizing Data

A data visualizer tab will not have any data plots up upon first being opened. To visualize the data which a tab has open, you need to click the Plot Data button found in the lower-right corner as the second button from the top in the data action button group. Once pressed, a new window will open; these matplotlib windows are where the plotted data will be displayed. The plots will take some time to be drawn, so please wait if the window initially opens up empty. The graphs will need to be redrawn if the window is resized.

Applying Filters

The list of available filters is found in the third list box from the left, towards the middle-right of the screen. This is a selectable list, and clicking on any filter will bring up more information on it.

Selecting a filter will do two things: display informational text below the filter list, and display parameter information to the right of the filter list.

The informational text for each filter gives a small overview of what the filter does when applied to the data. It is by no means sufficient to teach someone what applying the filter does; however, it does act as a great reminder for what filters do and is helpful for keeping track of all the operations being done on a set of data.

The parameter information is different filter to filter, and serves as an input for the functions that do require some sort of argument to run. Not every filter has user input, but those that do will bring up textboxes to be filled with the required parameters.

The wavelets filter acts a little bit differently than the others. There are many kinds of wavelet functions which can be applied, so the informational section is shorter and provides a like for further descriptions and the parameter section becomes another selectable list. This filter requires that the user selects the exact form of wavelet to apply, rather than filling in parameters.

Once a filter has been selected, and any extra information has been filled in, the filter must actually be applied. To apply a selected filter to the data, click the apply button right below the parameter section. The current filter will be moved into the Active Filters list alongside a short description of the parameters of it. You can also click the cancel button to clear the currently selected filter.

The final step to applying filters is to apply the active filters to the data. To do this, click the first button in the data operations group labelled Apply Filters. The active filters will be applied in order from the first on the list to the last. This process could take a significant amount of time depending on how many filters need to be applied and to how many datasets they are being applied. Give the application ample time to compute the resulting data. Should the application hang and fail to apply the filters, report the issue along with the active filter, their parameters, and the datasets being operated on. If you have changed the active filters on the current tab without applying them, then the active filters list may be inaccurate. To keep the filters accurate, always click the Reset Data button towards the bottom of the data operations group when you want to work with the clean data set.

Removing Filters

Filters can be removed from the list of active filters rather simply. Simply select the desired filter for removal from the active list and click the Remove button found below the list. Remember to reset the data and apply the filters again to get a new accurate plot.

Exporting Data

The application can export data in one of two formats, .PNG or .CSV, at any time after importing data. It will export the most recent transformation of the data, including the base data read in before any filtering. To access the export menu, click the Export Data button at the bottom of the data operations group.

A new window will open overtop of the application asking for both a filename and a file format to export the data to. You can type a name into the textbox, no need to include format in that name, and you can select the file type via the dropdown menu. You can hit the okay button to continue, or you can hit cancel to cancel the export at any time.

Once you hit the okay button, a file explorer window will open and ask you to choose a directory. Use this window to navigate to where you would like to save the file. Once you have found where you would like it, you can hit choose folder to save the file. You can also cancel at any time from this window using the cancel button.

Architecture

Leveraged Technologies and Packages

Anaconda

A popular distribution of Python used for data science and other scientific computing applications. Anaconda simplifies package management and is the base environment for this application.

Read more about Anaconda here: https://www.anaconda.com/

PyQT5

A set of Python bindings for the Qt cross platform application toolkit. That is to say, a Python implementation of one of the most popular libraries for UI design. The UI design and management for this application is all done through PyQt5 and the PyQt5 Designer.

Read more about PyQT5 here: https://www.riverbankcomputing.com/static/Docs/PyQt5/

Readgssi(Mention dependencies)

Readgssi is a GNU-licensed software hosted on GitHub made to read and process the data from GSSI GPR machines. The application is run via the command line, and takes some technical knowledge to run as a standalone application. Many features have been imported from readgssi and used in this application.

Read more about readgssi here: https://readgssi.readthedocs.io/en/latest/

Pandas

Pandas is a Python library that offers several useful data structures and operations for manipulating numerical tables and time series. This project leverages the DataFrame object and associated functions in particular.

Read more about Pandas here: https://pandas.pydata.org/

Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is used in this project for the data visualizations created from the data sets.

Read more about Matplotlib here: https://matplotlib.org/

Numpy

Numpy is a Python package developed to bring powerful computing components of older languages into the modern and easier to learn Python syntax. It is a powerful tool for calculation on it's own, and is a dependency for packages like Matplotlib and Scipy

Read more about Numpy here: https://numpy.org/

Scipy

Scipy is a Python ecosystem designed for mathematical and scientific computing purposes. It makes use of other packages such as Numpy and Matplotlib, but also comes with it's own features not found in the previous libraries.

Read more about Scipy here: https://www.scipy.org/

PyEMD

PyEMD is a Python module which implements the Empirical Mode Decomposition, which when combined with the Hilbert transform results in the Hilbert-Huang transform.

Read more about PyEMD here:

https://buildmedia.readthedocs.org/media/pdf/pyemd/latest/pyemd.pdf

PyWavelets

PyWavelets is an open source software implementing various wavelet transforms into Python.

Read more about PyWavelets here: https://pywavelets.readthedocs.io/en/latest/

Environment Folder Details

The folder that contains the application holds more than just the python code written for this tool, it includes all of the files necessary to build the environment in which the code can be run. Be sure to keep this environment unchanged unless you are patching in updates by hand.

The pychache folder contains compiled bytecode to speed up program start time. This folder and the files within will be created upon first running the program if they are not included in the environment in the first place.

The gprenv folder contains all of the environment packages required by the program, as specified by the datavis.yaml file. This is the folder being created and filled when the installer script is run.

The text files, license.txt and README.md, are both openable in basic text editors despite the different file format.The README file will have some basic details and instructions for the install and usage. The license file contains the license under which this application was developed, in particular the GNU GPL V3.

The .py files are the python files containing the code that runs the application. The file called dzt_visualizer.py is the main driver file. The file called backend almost all of the backend code, the logical connections between visual pieces of the software. Finally, popupWindows is the file which defines and builds windows like the export dialogue.

There are two image type files in the folder: loader.gif and favicon.ico. The favicon file helps to create the desktop icon, toolbar icon, and window icon. The loader.gif file is used in the program inside of the window which appears when a process may take some time to complete, and acts as a loading bar.

The final type of file in the folder are the scripts. The folder will initially only have one script: Install_GPR_Visualizer.bat. This is the script which, when run, will install the environment and make the code runnable. In that process, it creates three more scripts.

These scripts, Reinstall_Shortcut.bat, run.vbs, and dzt_visualizer.bat, automate basic operations on the application files. Run.vbs and dzt_visualizer.bat are the scripts which open the application. When the icon on the desktop is clicked, it targets run.vbs. This visual basic script then runs the dzt_visualizer batch file which finally opens the application. Reinstall_Shortcut.bat can be run to fix the desktop shortcut should it be lost or lose functionality.

Data Flows

General Program Flow

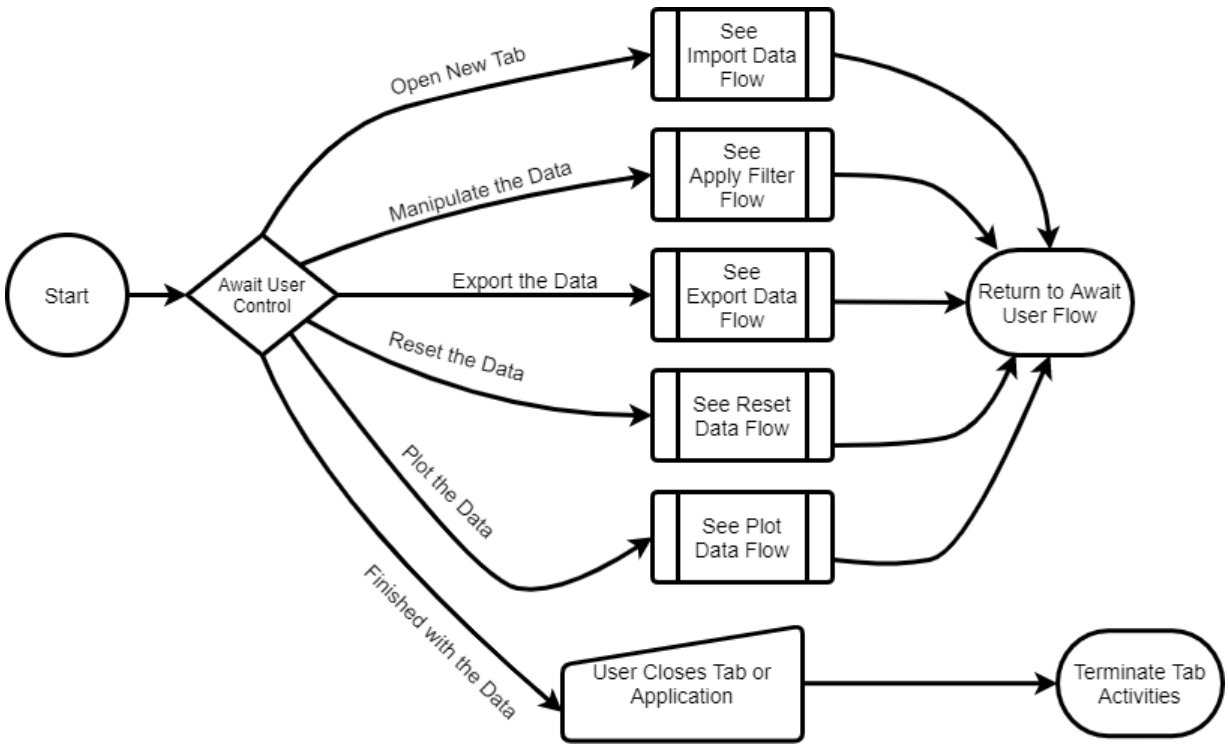

Figure 2: A data flow referencing how the program generally flows, mostly references other data flows

Import

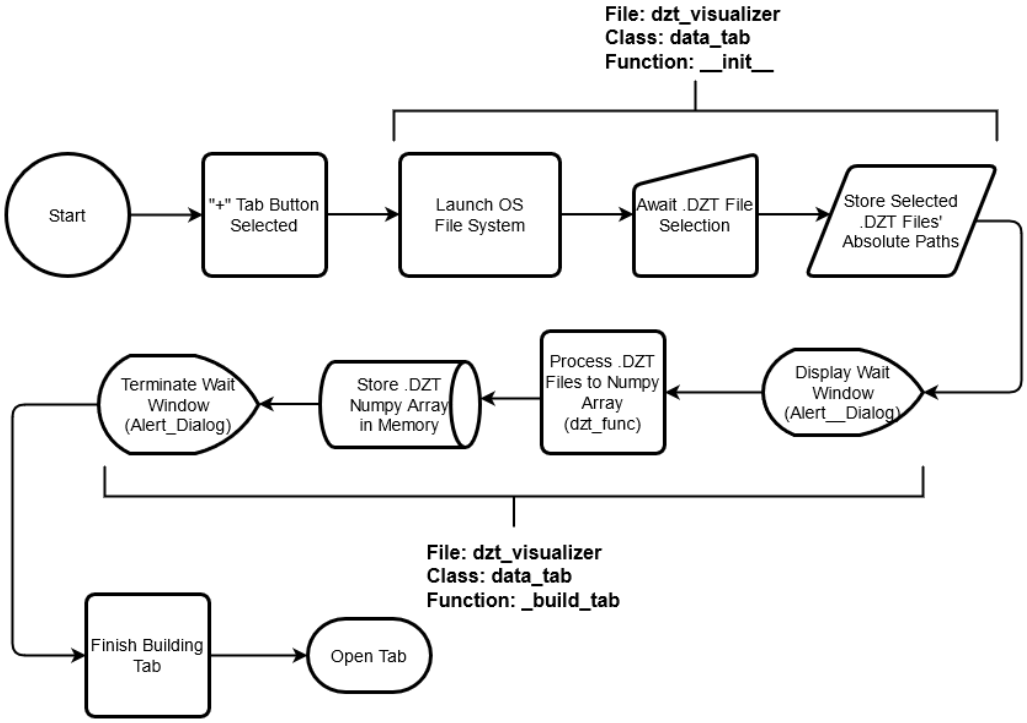

Figure 3: A data flow diagram detailing the data import/new tab functionality

Apply Filters

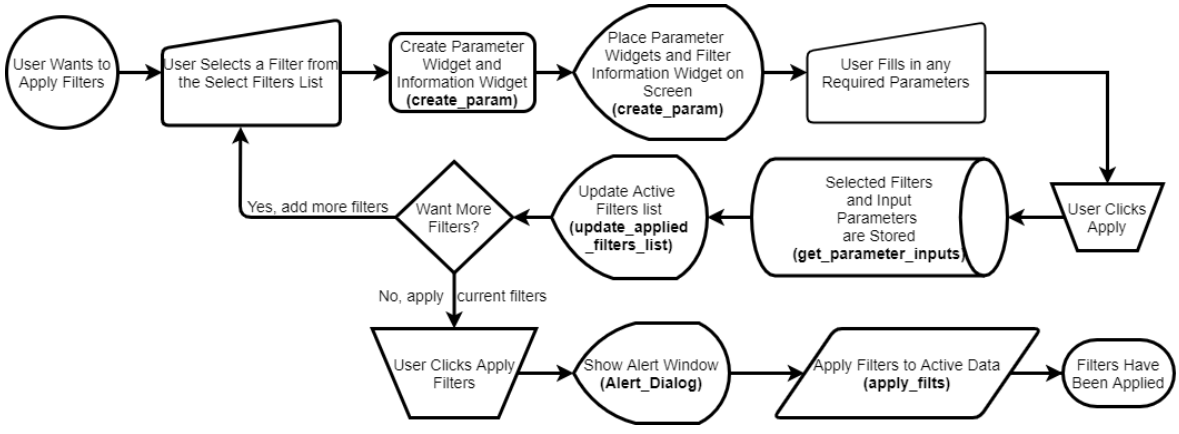

Figure 4: A data flow diagram detailing the filter application functionality

Plot Data

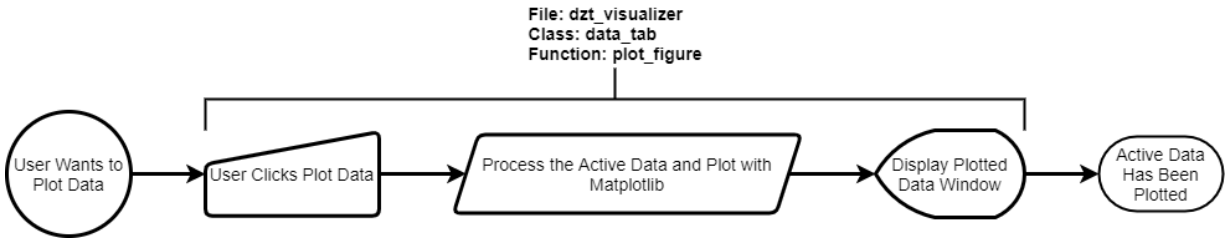

Figure 5: A data flow diagram detailing the data plotting functionality

Reset Data

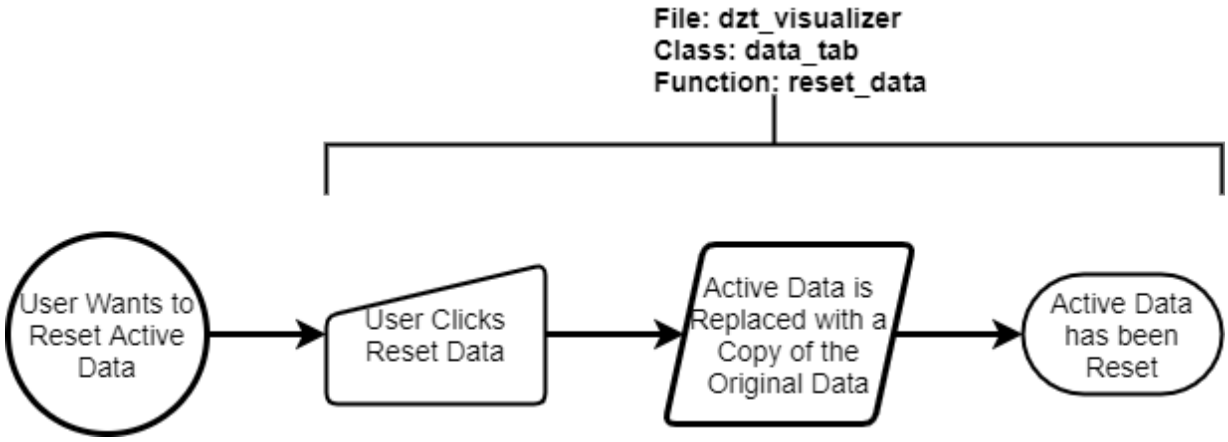

Figure 6: A data flow diagram detailing the reset data functionality

Export

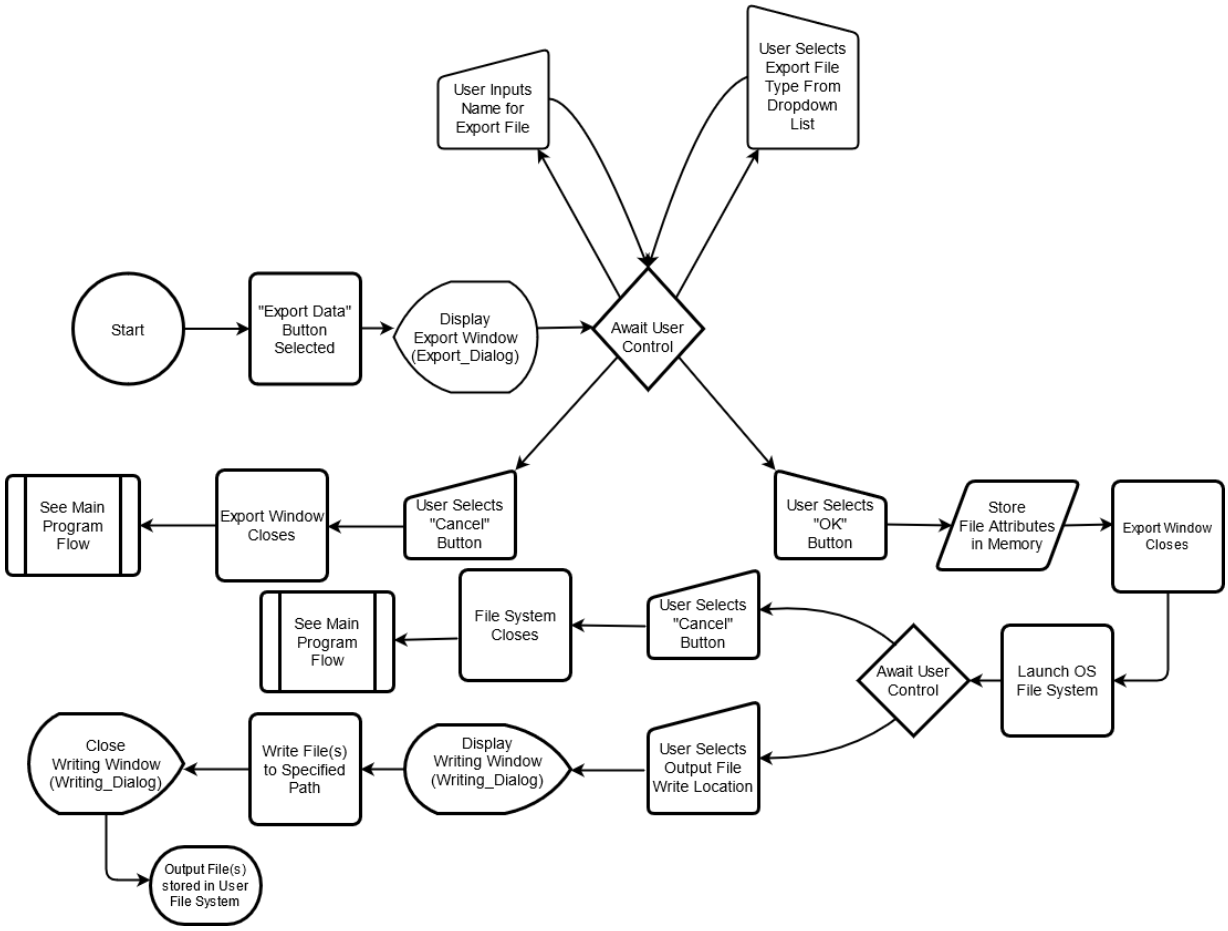

Figure 7: A data flow diagram detailing the data export functionality

Data Transform Implementation

Transforms Included

Horizontal Background Filter

The Horizontal Background Filter is implemented via code from readgssi. That implementation is found in the bgr() function in backend.py.

Vertical Triangular FIR Bandpass

The Vertical Triangular FIR Bandpass is implemented via code from readgssi. That implementation is found in the triangular() function in backend.py.

Fast Fourier Transform

The Fast Fourier Transform is implemented using the implementation present in scipy. In particular, the scipyfft.fft2() function is used.

Hilbert-Huang Transform

The Hilbert-Huang Transform is implemented using a Python module called pyhht.

Wavelet Transforms

The various Wavelet transforms implemented into this application are done so using a software called PyWavelets.

Location in the Code

The transforms are all pieces of the different mathematical packages which are included in the project. Find the specific package for each filter in its description above. Seeking the documentation for those specific functions of those packages will grant further insight into the implementation of each filter. The filters coming from readgssi are defined as functions within the code, but still come directly from readgssi.

Execution

The filters are all executed in the application in the backend.py file. You will find one section from those functions from mainstream packages, and one section for those coming from readgssi.

There is some interaction with the filters in the main driver file; however, this is all the logical and graphical elements of the filter selection system, and not the mathematical execution of those filters. Find this code in the dzt_visualizer.py file inside of the data_tab class.

Output

The output for the application is all handled in the export_pressed() function of the driver.

PNG Files

The image output for the application is given in the .PNG format. The image is simply the matplotlib graph of the current data saved via the saveplot() function native to the matplotlib package.

CSV Files

The raw data output for the application is given in the .CSV format. The csv files are simply created using the to_csv() function native to the pandas package.

Future Development

Risk Assessment

This application has been developed with an end goal of supporting risk assessment. One of the goals of the research which will make use of this tool is to uncover whether the known relationship between ballast size and rail integrity is useful in risk assessment. As the research uncovers more details surrounding the relationship, the risk assessment capabilities of the application can be expanded and fine tuned.

Colormapping is a good feature to leverage when building the risk assessment functionality. Matplotlib allows for colormaps to be defined in the code and used over any plot. So, mathematically define the risk and you have your color coded risk assessment.

At current, the application has no custom colormapping or risk assessment feature. A basic version of this could be created by defining a colormap to use hard coded or user given values to hard threshold the risk. This could grant some basic insight and be a good jumping off point for the research. Basically, plugging in low, medium, and high risk value thresholds into a colormap in the desired fashion.

Further down the line, there are other potential risk assessment options. One such option is to compile the results of multiple graphs into one holistic graph. This does not make sense for a lot of cases, but for the rail risk assessment there is potentially a case for this. Consider a train which carries three GPR sensors, one for the left, right, and middle of the track. These readings could potentially be compiled in some way to give a single graph for the entire rail.

Another option would be to implement three dimensional graphs into the project. Consider the option of seeing the three rail readings side-by-side in a three dimensional plot. This could begin to really paint the picture of what the rails look like out in the field.

All of this risk assessment will depend on what the research finds, and much will need to be done to create an accurate assessment model; however, this application is a great starting point for getting there.

Distribution Scope

The scope of the distribution of the application is another aspect where future work will need to be done. Currently, the installation process only works on Windows machines. This is because the process relies on batch scripts, or sets of commands written to be run by the command line. To Increase the installation scope, either more scripts must be written or a new installation model must be pursued.

The current installation files are written as batch scripts. This is a result of all of the team's main development being completed on Windows machines. In order for a similar

installation to be completed on other operating systems, new scripts must be written. Translating the current scripts into bash scripts would cover any *nix system, and only require one translation; however, they could be translated into any shell scripting language desired.

The team considered other avenues of installation as the development reached its later stages. One such avenue was creating an executable file via pyinstaller; however, no member of the team could get the tool to create a working file. This, to the best of our knowledge, is an issue with the dependencies. Should a future development team seek a new installation method, pyinstaller would be one avenue to avoid.

Seeking another avenue of installation beyond the scripts, especially if it yields an easier to operate installation, is one worthy future work. Any future work in this area would do well to keep in mind the dependencies of the project, especially the requirement of operating in an Anaconda environment.

Multi-Channel DZT Conversion

Currently, the application assumes that all input files were collected using, and represent, only a single channel. GPR machines come with the functionality to collect data on multiple channels. Where the program begins to read is defined by the header aliased by the statement header["rh_zero"]; the team currently assumes that this is where the first channel of the file begins. The rest of the channel start locations can be derived using the data offset found by multiplying the values found in the statements header["rh_nchan"]

and header[“rh_nsamp”] . In order to properly read a multi-channel file, there would need to be several changes put in place.

The main change which would need to be implemented is the functionality of readgssi to detect how a multi-channel file is partitioned. Currently, the function which would be used to properly read such a file, readgssi.dzt.arraylists(), is not in operation. This is because the current functionality fails to verify the channel offset values. As a result, attempts to run this function cause crashes in the readgssi code, resulting in a terminal error. The development team has encountered this, but has not made changes to the function due to a lack of information on the multi-channel features of GPR machines.

Should there be a need or a desire to implement multi-channel functionality in the future, there would need to be research done into the GPR machines being used by the research team in order to understand how the channels in the output data are split. Different GPR machines may operate with different data splits, and differing numbers of channels in use by the same machine may also operate under different data splits. These splits would need to be identified either by the documentation on the machines themselves, or by heavy experimentation. Once these splits are identified, then multi-channel files could be properly partitioned by readgssi.readdzt.arraylists() through hard coding in or getting user input values to pass to the function via parameters. No matter how it is implemented, there would need to be some heavy edits elsewhere in the code as well to truly plug this functionality into the application.

Code Appendix

help_tab

The help_tab class manages the creation and placement of the Help Tab. Its functions are as follows:

init()

This function runs when an instance of help_tab is created and builds the Help Tab.

data_tab

The data_tab class manages the main type of tab used in the application for storing data and acting as the user interface. Its functions are as follows:

init()

This function runs when an instance of data_tab is created, so when a user clicks the new tab button, and gets user input for it. In this case, the user input is the set of files selected from the file explorer upon tab creation. This function runs build_tab() with that data.

build_tab()

This function builds the tab according to the user data read in from init(). It uses the list of files to populate the Data Files list.

plot_figure()

This function runs when the Plot Data button is clicked. It actually loops through all the current data files and plots them in matplotlib.

apply_filts()

This function runs when the Apply Filters button is clicked. It actually applies the current list of filters to the current active data sets.

reset_data()

This function runs when the Reset Data button is clicked. It changes the current active data to what it was before any filtering was done on it. This allows users to change the filters run on a data set without creating a new tab.

remove_filter()

This function runs when the Remove button is clicked with an active filter selected. It removes the selected filter from the list of active filters.

create_param()

This function runs when the selected filter in the filter list changes. It creates the widget group to be displayed based on predefined setups. It also instantiates the needed parameter storage variables.

get_parameter_inputs()

This function runs after the create_param() function, and basically follows up on the work that it does. The parameter group defined before will be displayed, and the program will be ready to get the user input from the placed parameter inputs.

on_cancel()

This function runs when the cancel button below the parameter description is clicked. It clears the parameter section and filter info section of the current data.

update_applied_filters_list()

This function runs when the apply button below the parameter description is clicked. It stores the parameter input data and adds the filter to the active filters list.

export_pressed()

This function runs when the Export Data button is clicked. It brings up the export data window to take user input on the export, and then does the actual data writing based on those inputs and the current active data.

tab_manager

The tab_manager class is used as a parent widget to manage the other tab classes for dynamic tab capabilities. Its functions are as follows:

init()

This function runs when a new instance of a tab manager is created. It instantiates the widget and then runs the build_tabs() function.

build_tabs()

This function sets up the tab system at the top of the page, initially adding in the Home Tab and the New Tab button.

add_tab()

This function runs when the New Tab button is clicked. It creates a new tab and inserts it into the tab manager

remove_tab()

This function runs when the Close Tab button at the bottom of any tab is clicked. It removes the tab from the tab manager.

Ui_MainWindow

The Ui_MainWindow class is the main container class of the application, it is a window which holds the tab manager and all of its elements. Its functions are as follows:

setupUi()

This function runs when an instance of Ui_MainWindow is created, this happens when the application is first opened. It instantiates and builds the main window that the application is working in.

Export_Dialog

The Export_Dialog class is a dialog box class created to get user input for the export process. Its functions are as follows:

init()

This function runs when an instance of Export_Dialog is created. It runs the setupUi() function.

setupUi()

This function builds the export dialog box and sets up the backend connections for the input pieces. It also runs the retranslateUi() function.

retranslateUi()

This function simply sets the name of several widgets as a part of the conversion from a PyQT5 designer file to a Python file.

Writing_Dialog

The Writing_Dialog class is a dialog box class created as a popup for freezing the rest of the application while data is being exported. Its functions are as follows:

init()

This function runs when an instance of a Writing_Dialog is created. It tells the system that a subwindow is about to open, and then runs the setupUi() function.

setupUi()

This function builds a dialog box for informing the user to please wait while the system writes some data.

Alert_Dialog

The Alert_Dialog class is a dialog box created as a popup for freezing the rest of the application for reasons besides data writing. Its functions are as follows:

init()

This function runs when an instance of a Writing_Dialog is created. It tells the system that a subwindow is about to open, and then runs the setupUi() function.

setupUi()

This function builds a dialog box for informing the user to please wait while the system finishes some process.

Extraneous Functions

These functions do not belong to a class and are used independently of one.

All are found in the backend.py file. These functions are:

dzt_func()

This function runs when a set of input files is read into the application. This function utilizes the file reading abilities of readgssi to create the data sets to operate on.

dzt_filters()

This function runs when the Apply Filters button is clicked, and handles the application of all of the filters.

bgr()

This function runs when the dzt_filters() function encounters a Horizontal Background Removal selection. It utilizes readgssi's libraries to apply the HBR filter to the data.

triangular()

This function runs when the dzt_filters() function encounters a Vertical Triangular FIR Bandpass selection. It utilizes readgssi's libraries to apply the FIR filter to the data.