Spectrum Analyzer Tool

Software Verification Document

Prepared By:

Chris Williams
George Vendeta
Seema Kumaran
Jeffrey Amakihe
Alexandro Pinion

Date: 11/19/2023
Department: Computing and Software Engineering
Course: SWE 7903 - Capstone
Professor: Dr. Parizi

Table of Contents

1.1. INTRODUCTION

1.1 Purpose

A Software Verification Document serves (SVD) as the documentation of the verification activities carried out during the development and testing of software. Verification is the process of evaluating a system or component to determine whether it meets the specified requirements. The primary goal of verification is to ensure that the software functions as intended and meets the quality standards set for it by the requirements. The intended audience for this SVD is for developers, Robins AFB stakeholders, testers and project managers.

1.2 Scope

The goal of this Software project is to create, test, and implement a user-friendly program that the engineers and analysts of Robins AFB can use to reduce manual toil and redundancy. This objective will be achieved by creating a video analysis tool, that can analyze videos of spectrum analyzer signal activity, we can extract the signal intercept data and reduce the time it takes to complete an analysis from as much as 8 hours to 5 minutes or less. In addition to developing the video analysis, an independent deployment environment for the client's platform, formatted CSV output data representing spectrum analyzer signals, and image signal processing are required. The project's goal is to provide a reliable and scalable solution that addresses the client's issue within the allocated spending limit and time frame.

1.3 Overview

This document will go over the system architecture, data flow, user interfaces, algorithms, data structures, security considerations, error handling, and other subjects are covered in detail in this document. It serves as an essential means of communication for development teams, testers, project managers, and stakeholders, guaranteeing that all parties participating in the project comprehend the functioning and design of the product. An integral part of the software development lifecycle, the SDD also acts as a point of reference for scalability and future maintenance.

1.4 Reference Material

C. Williams, “Spectrum Analyzer Software Requirements Document (Draft -SWE 7903 Spectrum Analyzer Capstone -Team 5),” Sep. 2023.

2. SYSTEM OVERVIEW

As a key aviation hub for the United States Air Force, Robins Air Force Base (AFB) uses Spectrum Analyzers to measure the quality of transmission signals within their ground stations. These Analyzers gather information while in flight, producing hours' worth of crucial data that is necessary to identify signal intercepts resulting from RF threat

systems employed against Aircraft Electronic Warfare (EW) systems during flight. There are, however, operational inefficiencies and a significant human resource cost associated with the current manual procedure of examining video recordings of these videos to develop datasets of the anomalies in recorded signals. In response to this challenge, KSU students have been requested by Robins AFB to develop an automated system for the analysis of EW signal intercepts, and subsequently storing the processed data output in a comma separated value (CSV) format. This will guarantee prompt identification of problems with signal quality while optimizing operations and resource use. This group of students behind this project will create a standalone application that will operate on the computers that are intended for use by Robins AFB personnel. The application will offer a guided process for loading relevant data, processing it, and producing output resources that will be most helpful to the stakeholders at Robins AFB.

3. SYSTEM ANALYZER SOFTWARE VERIFICATION PLAN

3.1. Overview of verification strategy

To fully analyze and verify the spectrum analyzer software, the customer requirements will be utilized to ensure all features meet customer design standards. The method of verification in which will be employed is the process of dynamic verification. This involves experimentation, dynamic testing. This is an ideal for finding faults or bugs in the software.

3.2. Verification Test Plan

Requirement 5.5 - Create a Graphical User Interface (GUI) to allow user to interact with program.

- Test Plan: We are going to create a GUI using PyQT5 to allow the user to interact with our program. This will allow the user to set calibration parameters, file storage location, and file selection for processing.
- Test Case: We created a GUI to allow the user to interact with the program. This is important as the user input parameters are important for our program to run correctly. For this we used PyQT5 as supposed to Kivy for the ease of use from a developer and user perspective. Our test consisted of launching our GUI using Python 3.2 shell and Pycharm. We verified that the user is able to perform this action without any hiccups.

Requirement 5.6 - 8 Hour Video Processing time less than 5 minutes:

- Test Plan: We are going to calibrate our program and discern baseline parameters that will comply with this requirement. Parameters to baseline are as follows; RGB values, Framerates, template image snippet.
- Test Case: To satisfy this requirement was verified by calibrating the program using the GUI calibration page. In this page, we played around with the RGB values and settled on the one that worked the best for general usage and set these values as default values in the _ini_ file. The framerate was also chosen to be adequate so that the output is not dropping frames, but it is also not overwhelming local storage and processing power. The user does have the ability to increase the output parameters for higher

fidelity; however, this will take longer processing time and may not comply with the 5-minute requirement.

Requirement 5.7 - Deploy on a Windows 10 physical server from a CD/DVD (not Blu-ray)

- Test Plan: We are creating a deployable API so that the user could download the Spectrum Analyzer Program onto a portable drive which they can deploy in various lab settings.
- Test Case: To verify this requirement, we had two of our team members download the program package individually and install it on their local Windows machines. This trial run presented some challenges as the Github repository didn't branch correctly and some of the program files were corrupted. However, we sorted those issues out and were able to download it onto our team members' machines and run it from there just to ensure that our API is easily deployable.

Requirement 5.8 - Follow industry secure coding best practices.

- Test Plan: We are planning to package our program into an API that could be used offline. This includes having the user download the program onto the local drive and not depend on internet connection or cloud services. Our program is meant to be highly portable and memory/processing power efficient. The key to this requirement is to avoid using the cloud services or drivers that need constant over the air updates.
- Test Case: To verify this requirement, we downloaded the program onto a flash drive memory stick and installed it onto our offline machines. Our program is meant to be installed onto local drive and we wanted to ensure that this wouldn't be an issue. Our program should run just fine without usage of internet connection or updating dependency drivers.

3.3. Roles and responsibilities of the verification team

The Verification Team lead is George Vendeta, his role includes Overseeing the entire verification process. He also acts a s a primary point of contact between the verification team and multiple stakeholders. His responsibilities include developing the overall verification strategy and plan, allocating tasks to team members based on their individual expertise. He also monitors progress, and addresses issues to ensure verification goals are met.

The Test Manager is Chris Williams, his role includes day to day testing activities, he coordinates testing efforts and resources. His primary responsibilities include developing a detailed test plan based on the verification strategy, managing the testing schedule and resource allocation. Lastly Tracking and reporting testing progress.

The Test Engineer is Alex Pinion, his role includes performing the actual testing activities and creating and execute test cases. His responsibilities include developing detailed test cases based on requirements and specifications,

executing test cases manually or using automated testing tools.

The Security Testing Engineer is Seema Vayaliparambil Kumaran. Her role includes identifying and addressing security vulnerabilities in the software. Her responsibilities include conducting security assessment and penetration testing, identifying security weaknesses and recommending remediation.

The Performance Testing Engineer is Jeffrey Amakihe. His role includes assessing the performance characteristics of the system. His responsibilities include executing performance testing scenarios, analyzing system performance metrics and identifying areas of improvement.

3.4. Schedule and milestone for verification activities

Date (YYYY-MM-DD)	Milestone/ task	Deliverable	Phase
2023-11-13	Testing	Test Level 1 Requirements	Verification
2023-11-13	Milestone 1	Test Cases passing at 100%	Verification
2023-11-14	Preliminary Code	Code version that may be incomplete yet functional for review	Monitoring and Controlling
2023-11-22	Final Code	Video of process and code working along with appropriate background, processes, and outcomes	Closure

4. SYSTEM ARCHITECTURE

4.1. Architectural Design

This section includes a high-level structure design that outlines the main elements, how they relate to one another, and how the application is organized overall. It acts as a construction blueprint for the software, assisting developers in comprehending how different system components interact to produce the intended functionality. Important non-functional needs like performance, maintainability, and scalability are covered. This section will also provide a high-level road map that directs the development team in creating a system that satisfies the project's functional and quality objectives while taking flexibility and maintenance ease into account is the aim of architectural design. For the Spectrum Analyzer Tool application to remain viable and successful over the long term, a well-designed architectural design is essential. Below are the main components of the systems architecture:

4.2. User Interface (UI) Layer:

4.2.1. Layout of design

For the spectrum analyzer, a user is provisioned with a simple UI, where user can select the desired video file for processing, select the calibration values for processing, and save the output csv file in the desired location.

1. File Uploader Window:

In this screen, a user can select the input file location where the video file is available. It can be in mpeg format. The location where the user wants to save the

output csv file can also be selected from this UI. Once the location is finalized, the user can navigate to the next screen.

2. Calibration Window:

This screen allows users to proceed with the default High and Low values of RGB for the video processing. If the user wants to change the RGB values of High parameters, user can click on 'Edit' button and select the color from the color picker. The UI get reflected with the RGB values of selected color.

Users can edit RGB values of Low parameters as well.

3. Video Processing Window:

This window waits for completion of processing and progress percentage being shown to customer real time. Once the processing is over, the user can stick to the location selected for the output or he/she can change the location by going back to the initial screen. Once the location is finalized, the output file gets saved in the selected location. Now the user can close the window.

4.2.2. Logic Layer:

4.2.2.1. Video Processor:

This part extracts pertinent data by processing the loaded video frames. It could involve filtering, feature extraction, picture analysis, and other image processing methods. The data has been processed and is ready to be converted to CSV format. This layer will perform the business logic for the application.

4.2.3. Data Management Layer:

4.2.3.1. CSV Data Converter:

This module creates a CSV file from the processed video data. It arranges the information in a tabular style that can be exported as CSV. The CSV format may be configured by users with options to set headers and delimiters, for example. This layer will also be responsible for creating all the non-volatile data storage for the application.

4.2.4. Exceptions and Logging Layer:

The exceptions and logging layer will put in place an error-handling system to handle unforeseen problems that arise when loading, processing, or converting videos. This mechanism will produce logs for reporting errors and debugging.

4.2.5. Distribution Layer:

Builds all packages and bundles all preset libraries, dependencies, and components that makes it simpler to deploy the application to the user. Generates a standalone executable that operates independently on a computer system without the requirement for extra resources or dependencies.

The Figure 1 below showcases an overview diagram of the architectural design for the Spectrum Analyzer Tool and how each of the components in the application provide key

functionality. Each of the diagram's components are described above.

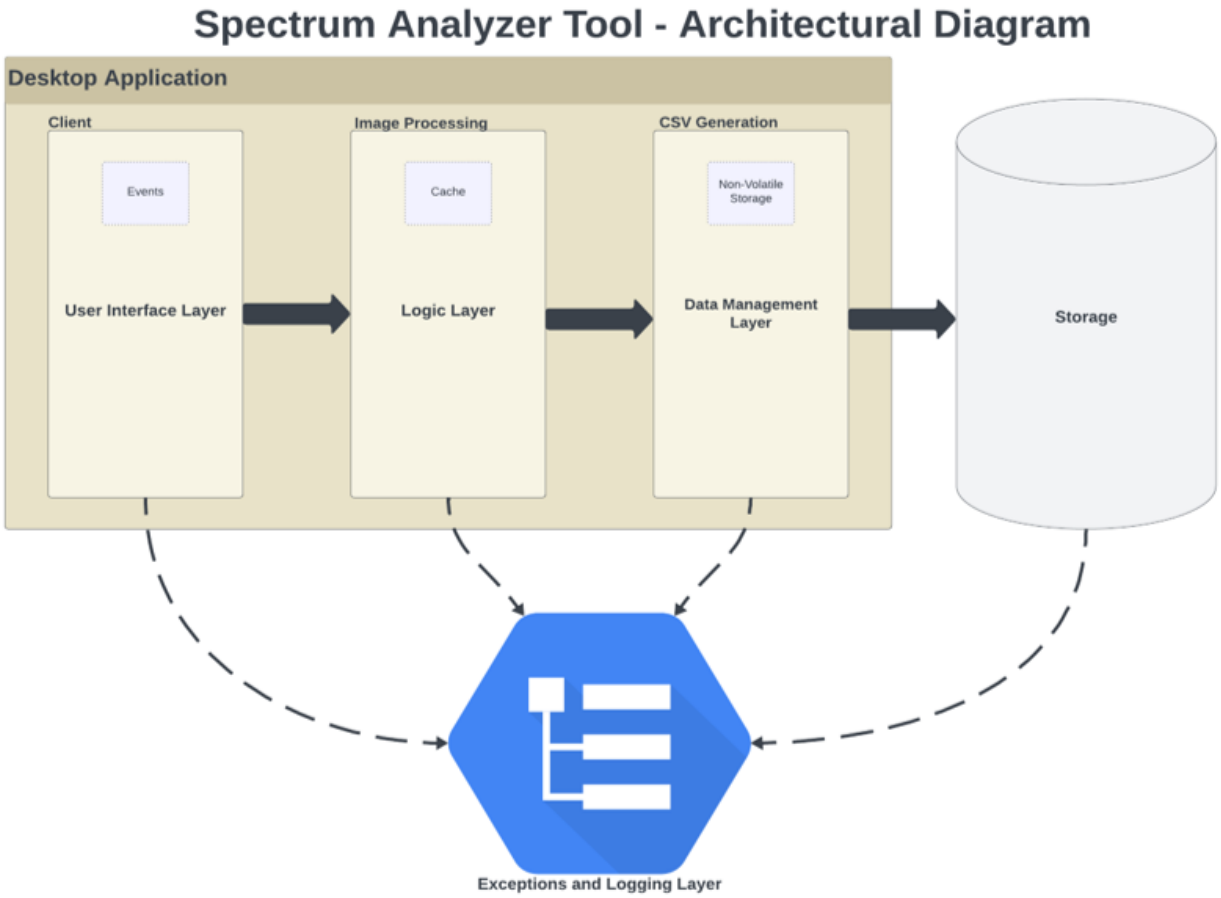


Figure 1 - Architectural Diagram of Spectrum Analyzer Tool

5. SYSTEM FEATURES

5.1. System Feature – User Input

5.1.1. Description and Priority

Priority: 1

The range and allowed min/max frequency and amplitude information that are used to analyze the video differ from test to test. To adjust accordingly, those settings must be noted at the beginning of the program before the video is analyzed.

5.1.2. Stimulus/Response Sequences

Scenario 1: Valid user input

- Stimulus: All fields are populated by the user and in proper format.
- Response: Video analysis starts and continues to generate appropriate output.

Scenario 2: Incomplete user input.

- Stimulus: One or several fields are not populated by the user before starting analysis.
- Response: If a single field is missing data, that field will be referenced in an error message. If multiple fields are missing user input, a generic message stating that will be output.

Scenario 3: Invalid user input.

- Stimulus: One or several fields are populated with invalid data, e.g., alphanumeric instead of simple numeric data.
- Response: If a single field has invalid data, that field will be referenced in an error message. If multiple fields contain invalid data, a generic message stating that will be output.

5.1.3. Functional Requirements

Command line parameters or a UI prompt at program start is needed to identify the parameters that should be used to analyze the video.

5.2. System Features – Analyze Video of Spectrum Analyzer

5.2.1. Description and Priority

Priority: 1

Program must analyze and determine data from the video including the elements mentioned in 3.1. The analysis of the data can be based on 2 scenarios. The user may input the framing reference data that will be used to calculate the threshold, or the program may infer the threshold.

5.3. Functional Requirements

Scenario 1: Priority 1

The program will analyze the graph data on screen to determine the threshold value. When the power level exceeds that threshold level the program will calculate the data for the signal intercept.

Scenario 2: Priority Optional

The program will use the user input to analyze the graph data on screen. When power level exceeds the threshold level the program will calculate the data for the signal intercept

5.4. System Feature – CSV File Output

5.4.1. Description and Priority

Priority: 2

CSV file containing the information specified in 3.1.

5.5. Description and Priority

System Feature – Graphical Output

5.5.1. Description and Priority

Priority: Optional - 2

Provide a graphical representation of the data relayed in the generated CSV. Two levels of information, both summary and outliers represented.

5.6. Functional Requirements

5.6.1. The GUI should:

- represent the summary data in a graphical display.
- represent the outlier data in a separate visualization.
- provide the capability to forward the video to a particular outlier.

5.7. Performance Requirements

Process 8 hours of video in 5 minutes

5.8. Deployment Requirements

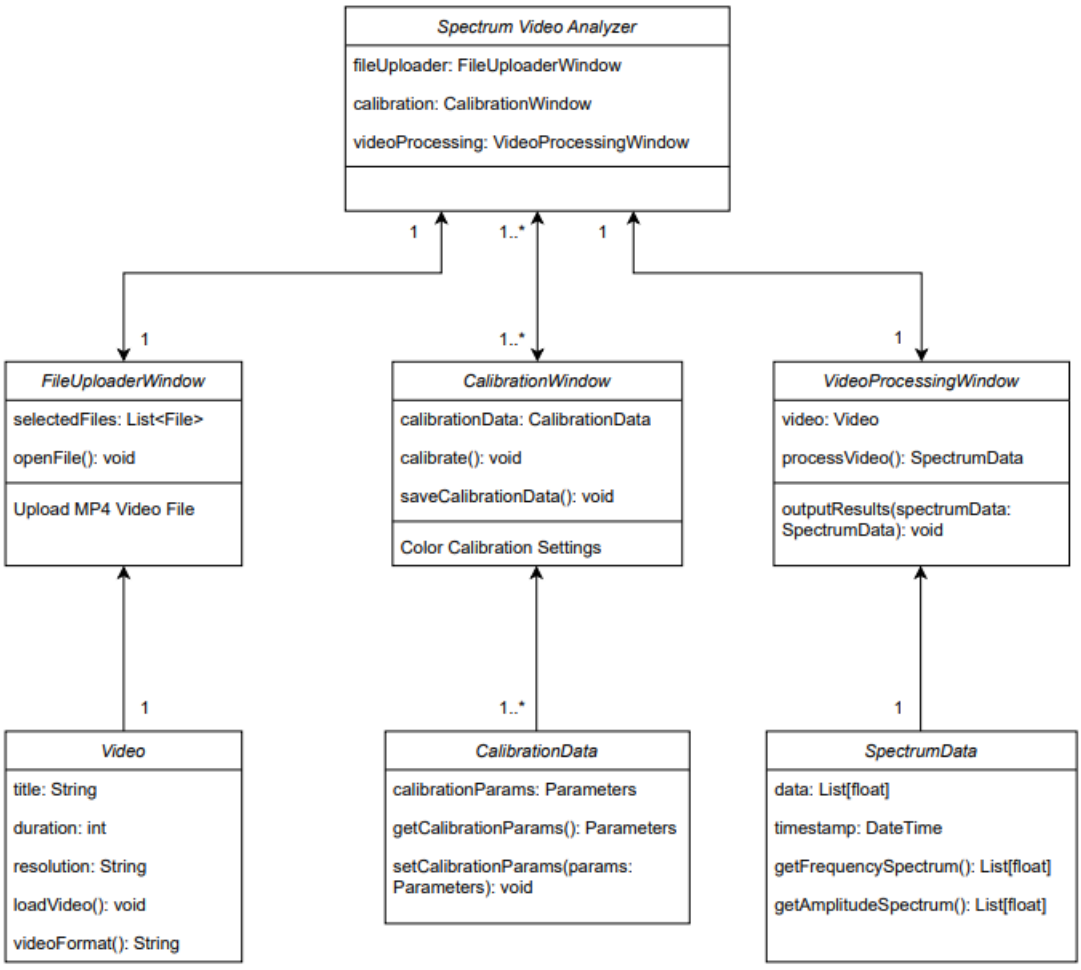
Deploy on a Windows 10 physical server from a CD/DVD (not Blu-ray).

5.9. Security Requirements

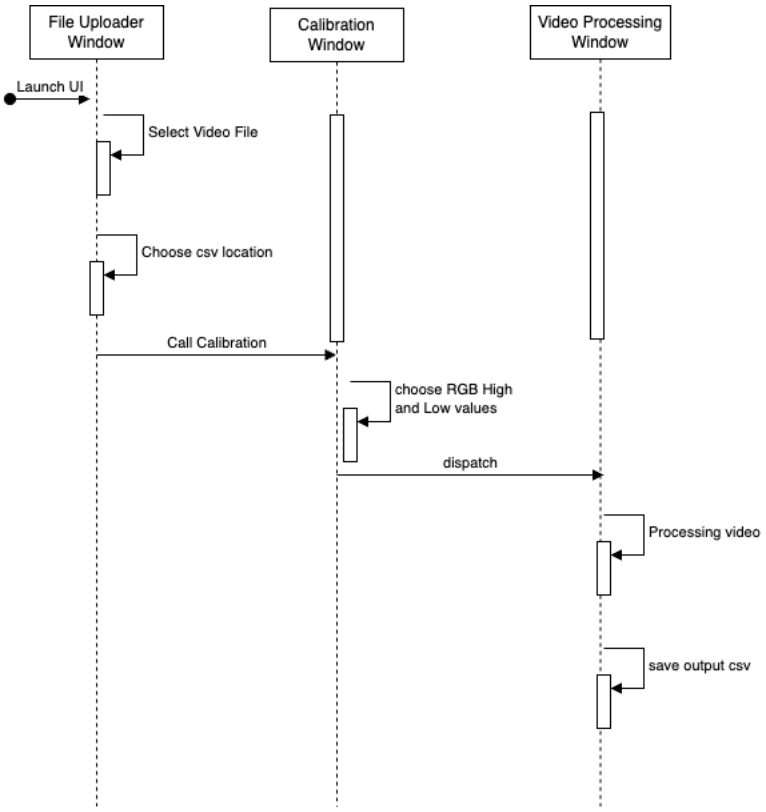
Follow industry secure coding best practices.

5.10. DETAILED DESIGN

5.10.1. Class Diagram



5.11. Sequence Diagram

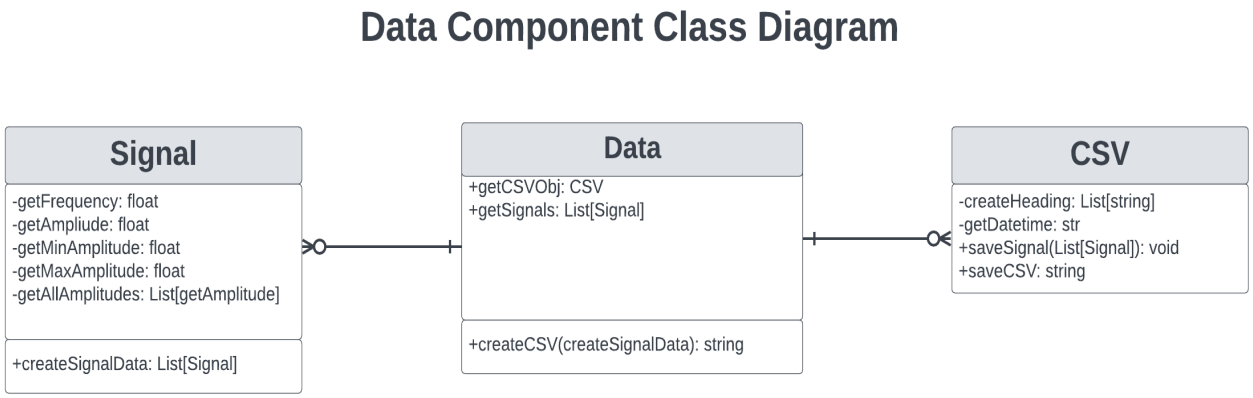


5.12. Pseudocode

In this class diagram, the main class is "Spectrum Video Analyzer," which has instances of the FileUploaderWindow, CalibrationWindow, and VideoProcessingWindow classes. FileUploaderWindow allows the user to select and upload files. CalibrationWindow handles calibration and stores calibration data in a CalibrationData object. VideoProcessingWindow processes video and displays results using a Video class and SpectrumData. The CalibrationData class holds calibration parameters, and the Video class represents the video being processed. The SpectrumData class holds the data related to spectrum analysis.
Note that this is a simplified class diagram. Depending on your specific requirements, you may need to add more attributes and methods to these classes.

5.13. DATA DESIGN

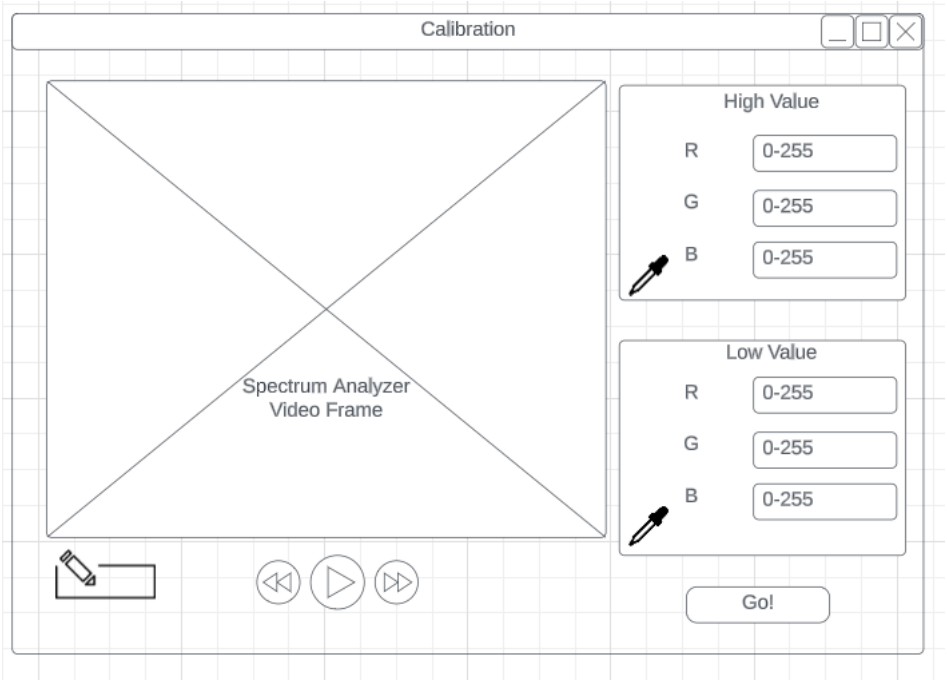
In this section we depict the Signal, Data and CSV classes. We have multiple functions which retrieves float data types of all the pertinent information within the Signal class. This information is then stored in an array list. The signal list is then called to the Data class, where a CSV object is then created. The CSV class then receives and stores the signal list which contains all the Amplitudes.



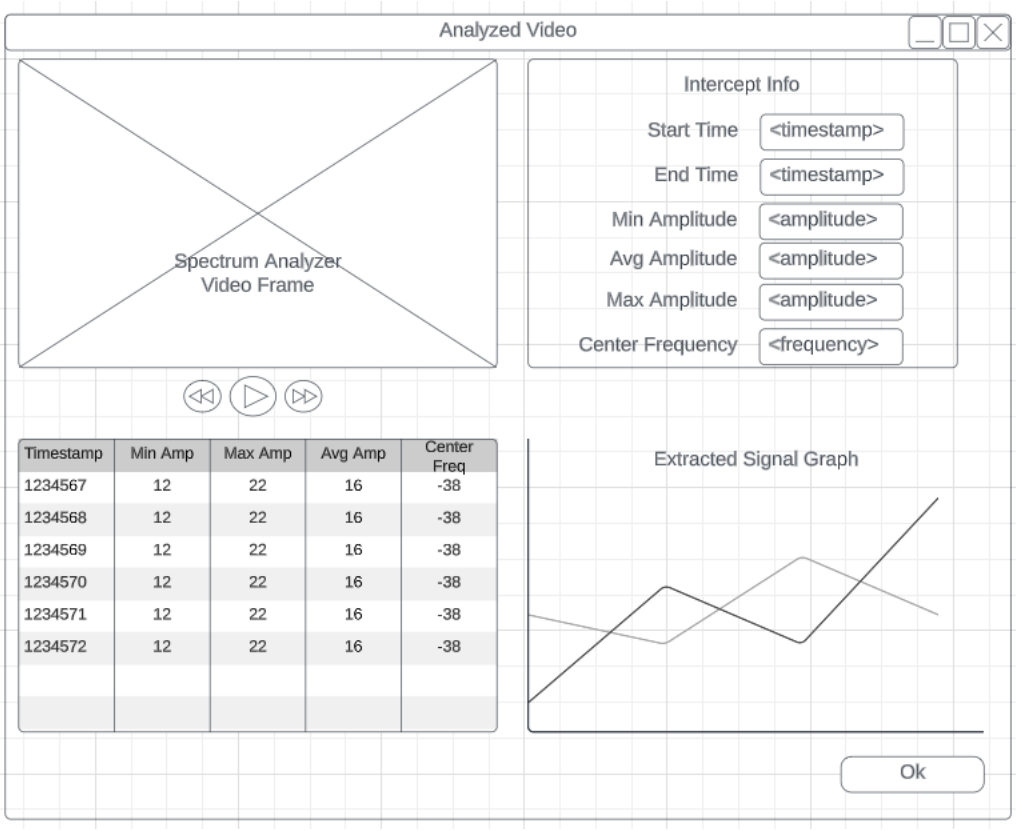
5.14. HUMAN INTERFACE DESIGN

5.14.1. UI design/wireframes

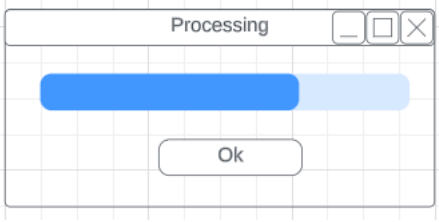
Calibration Window



Analyzed Video Output Window

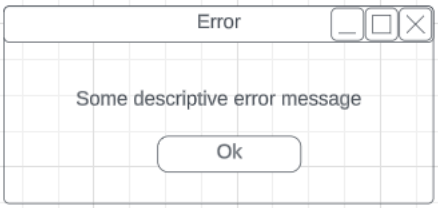


Progress Bar

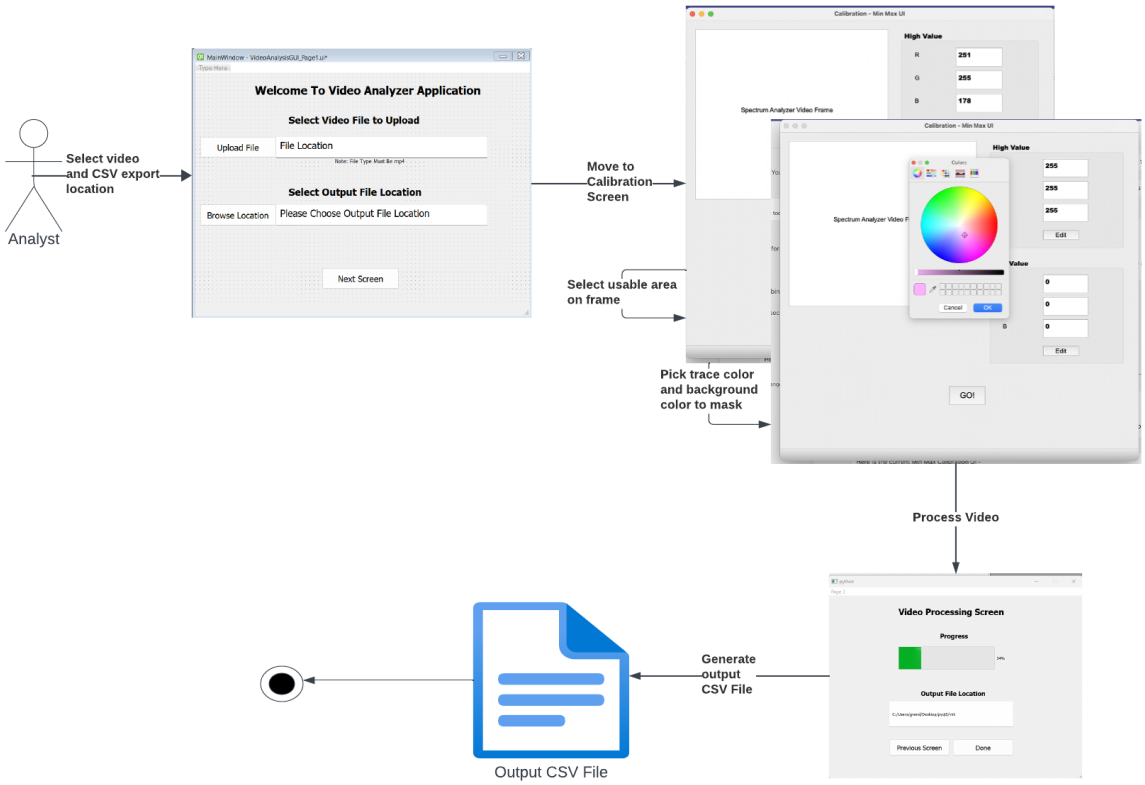


Data Input Dialog

Error/Warning Dialog



5.15. UX design



APPENDICES

Traceability Matrix

Req Id	Requirement	Design Feature	Section
3.1.1, 4.1	Input data requirements – Required	CLI and UI design	6.1
3.1.2.1, 4.3	Required Output – csv output	Data Management Layer	3.1.3
3.1.2.2	Optional Output – csv	User Interface Representation	3.1.1
4.2	Analyze video of signal intercepts	Logic Layer	3.1.2
5.2	Deployment	Distribution Layer	3.1.5