SigPro Vehicle Detection Application

Final Report

Stephen Norman, TJ Cyrus, Tomoki Nakamura, Senay Grima, Freddy Manyaka

CMSC 495: Capstone in Computer Science

Professor Shanna Nevarez

4 March 2024

# 

# 

# Table of Contents

[Table of Contents 2](#_Toc160286606)

[Project Team 3](#_Toc160286607)

[Overview (+ Rationale) 3](#_Toc160286608)

[Project Plan 5](#_Toc160286609)

[Documentation 5](#_Toc160286610)

[Phase I 5](#_Toc160286611)

[Phase II 5](#_Toc160286612)

[Phase III 6](#_Toc160286613)

[Final 6](#_Toc160286614)

[Requirements Specification 6](#_Toc160286615)

[System Specifications 6](#_Toc160286616)

[● Development Software 6](#_Toc160286617)

[● Development System Specifications 8](#_Toc160286618)

[● Operating System Specifications 8](#_Toc160286619)

[User's Guide 8](#_Toc160286620)

[Introduction 8](#_Toc160286621)

[System Requirements 9](#_Toc160286622)

[Installation Instructions 9](#_Toc160286623)

[User Interface Overview 10](#_Toc160286624)

[Using SVDA 11](#_Toc160286625)

[Frequently Asked Questions & Troubleshooting 12](#_Toc160286626)

[Additional Support 13](#_Toc160286627)

[Test Plan (& Results) 13](#_Toc160286628)

[Introduction 13](#_Toc160286629)

[Test Items 14](#_Toc160286630)

[Test Case Specifications 14](#_Toc160286631)

[Test Report & Traceability Matrix 17](#_Toc160286632)

[Design 24](#_Toc160286633)

[UI Design 24](#_Toc160286634)

[Classes, Methods, Fields, & Interfaces 24](#_Toc160286635)

[**SVDA.py** 24](#_Toc160286636)

[**VehicleDetector.py** 25](#_Toc160286637)

[Data Structures, Data File Structures, Inputs, Outputs 26](#_Toc160286638)

[Approach & Performance Estimates 26](#_Toc160286639)

[Policy of Software 27](#_Toc160286640)

[Development History 27](#_Toc160286641)

[Phase 1 27](#_Toc160286642)

[Phase 2 29](#_Toc160286643)

[Phase 3 31](#_Toc160286644)

[Conclusion 33](#_Toc160286645)

# Project Team

|  |  |
| --- | --- |
| Stephen Norman | Team Lead |
| TJ Cyrus | Programmer |
| Tomoki Nakamura | Programmer |
| Senay Girma | UI Designer |
| Freddy Manyaka | UI Designer |

# 

# Overview (+ Rationale)

For this project, an application was created that performs object detection using a Haar Cascade Classifier to detect vehicles in a video feed. A Haar Cascade Classifier is a machine-learning model that detects objects in image or video files using simple feature detection. The application uses OpenCV to implement the classifier and process the video. Additionally, the application is written in Python - a high-level, general-purpose programming language that supports multiple programming paradigms.

Cities and interstates are constantly affected by traffic problems. By using cameras at intersections and streets, along with methods to detect vehicles, data can be gathered that can help reduce traffic jams. For instance, the data can be used to generate live traffic information that can help commuters choose different routes during peak hours. Another possibility is that city planners can use the data on the vehicle count at an intersection to decide where to build more infrastructure.

# 

# Project Plan

|  |  |  |
| --- | --- | --- |
| **Task** | **Due Date** | **Responsible Party** |
| Documentation | | |
| Project Plan Draft | 1/20/2024 | Stephen (Team Lead) |
| Project Plan Review | 1/22/2024 | All Members |
| Project Plan Final | 1/23/2024 | Stephen |
| User Guide Draft | 1/27/2024 | Senay (UI Designer) |
| I User Guide Review | 1/29/2024 | All Members |
| User Guide Final | 1/30/2024 | Senay |
| Test Plan Draft | 1/27/2024 | Freddy (UI Designer) |
| Test Plan Review | 1/29/2024 | All Members |
| Test Plan Final | 1/30/2024 | Freddy |
| Project Design Draft | 2/3/2024 | TJ, Tomoki (Programmers) |
| Project Design Review | 2/5/2024 | All Members |
| Project Design Final | 2/6/2024 | TJ, Tomoki |
| Phase 1 Report Draft | 2/11/2024 | Stephen |
| Phase 1 Report Review | 2/12/2024 | All Members |
| Phase 1 Report Final | 2/13/2024 | Stephen |
| Phase 2 Report Draft | 2/18/2024 | Stephen |
| Phase 2 Report Review | 2/19/2024 | All Members |
| Phase 2 Report Final | 2/20/2024 | Stephen |
| Phase 3 Report Draft | 2/25/2024 | Stephen |
| Phase 3 Report Review | 2/26/2024 | All Members |
| Phase 3 Report Final | 2/27/2024 | Stephen |
| Final Report Draft | 3/4/2024 | Stephen |
| Final Report Review | 3/5/2024 | All Members |
| Final Report Final | 3/6/2024 | Stephen |
| Phase I | | |
| Project Structure Setup | 2/7/2024 | TJ, Tomoki |
| Initial App that loads & displays an image | 2/10/2024 | TJ, Tomoki |
| Basic UI created | 2/10/2024 | Senay, Freddy |
| Train Haar Classifier | 2/12/2024 | Stephen |
| Phase II | | |
| Vehicles detected in image | 2/15/2024 | TJ, Tomoki |
| App loads and displays video from file | 2/17/2024 | TJ, Tomoki |
| App loads and displays video from stream | 2/18/2024 | TJ, Tomoki |
| First Draft of UI complete | 2/17/2024 | Senay, Freddy |
| Phase III | | |
| App Detects vehicles in video | 2/22/2024 | TJ, Tomoki |
| App logs detections information | 2/25/2024 | TJ, Tomoki |
| Compare our classifier with open-source classifier and choose the best one for the application. | 2/25/2024 | Stephen |
| Main UI Complete | 2/24/2024 | Senay, Freddy |
| Final | | |
| Implement processing logged data showing the number of vehicles per unit of time. | 3/4/2024 | Stephen, TJ |
| UI for visualizing the data complete | 3/4/2024 | Senay, Freddy |

# Requirements Specification

1. The application shall provide an easy-to-use user interface.
2. The application shall support mp4 and avi video.
3. The application shall support a M3U8 Stream format.
4. The application shall allow the user to select between a video file or a link to a stream.
5. The application shall handle errors and exceptions gracefully.
6. The application shall allow the user to start and stop detection.
7. The application shall display the video in an application window.
8. The application shall apply computer vision algorithms to detect vehicles in the video.
9. The application shall visually indicate the detection on the video.
10. The application shall log the detections at a minimum time and event-id will be recorded.
11. The application shall allow a user-selectable classifier.
12. The application shall show a graph of the number of vehicles detected over time.

# System Specifications

## Development Software

* + Python v3.12.1
  + The following dependencies were also used. See section: User Guide for installation instructions.
    - bleach==6.1.0
    - bokeh==3.3.4
    - certifi==2024.2.2
    - charset-normalizer==3.3.2
    - colorama==0.4.6
    - contourpy==1.2.0
    - customtkinter==5.2.2
    - cycler==0.12.1
    - darkdetect==0.8.0
    - fonttools==4.47.2
    - haar-cascade-nms==1.1.1
    - idna==3.6
    - imageio==2.33.0
    - imutils==0.5.4
    - Jinja2==3.1.3
    - kiwisolver==1.4.5
    - linkify-it-py==2.0.3
    - Markdown==3.5.2
    - markdown-it-py==3.0.0
    - MarkupSafe==2.1.5
    - matplotlib==3.8.3
    - mdit-py-plugins==0.4.0
    - mdurl==0.1.2
    - numpy==1.26.3
    - opencv-contrib-python==4.9.0.80
    - opencv-python==4.9.0.80
    - opencv-python-headless==4.9.0.80
    - packaging==23.2
    - pandas==2.2.0
    - panel==1.3.8
    - param==2.0.2
    - Pillow==10.1.0
    - pyparsing==3.1.1
    - python-dateutil==2.8.2
    - pytz==2024.1
    - pyviz\_comms==3.0.1
    - PyYAML==6.0.1
    - requests==2.31.0
    - six==1.16.0
    - tornado==6.4
    - tqdm==4.66.2
    - typing\_extensions==4.9.0
    - tzdata==2024.1
    - uc-micro-py==1.0.3
    - urllib3==2.2.1
    - webencodings==0.5.1
    - xyzservices==2023.10.1

## Development System Specifications

* + PC running Windows, MacOS or Linux
  + Intel or AMD CPU
  + 16 Gigabytes of memory or more

## Operating System Specifications

* + PC running Windows, MacOS or Linux
    - Windows 10 or newer
    - MacOs 10.15 (Catalina) or newer
  + Python 3.12.1 or later
  + Intel or AMD CPU
  + 16 Gigabytes of memory or more
  + Internet Connection

# User's Guide

## Introduction

SigPro Vehicle Detection Application (SVDA) is a software tool used to detect objects and perform analysis on a given video input. SVDA uses Haar Cascade Classifier to detect and track vehicles in car traffic videos. Data gathered from the video is used to generate live traffic information. This document is intended to provide the user with all the information needed to get started with and use the SVDA software effectively.

## System Requirements

Before installing SVDA software, please make sure your computer meets the following requirements:

* Operating System Requirements:

o Windows 10 or newer

o MacOS 10.15 (Catalina) or newer

o Linux

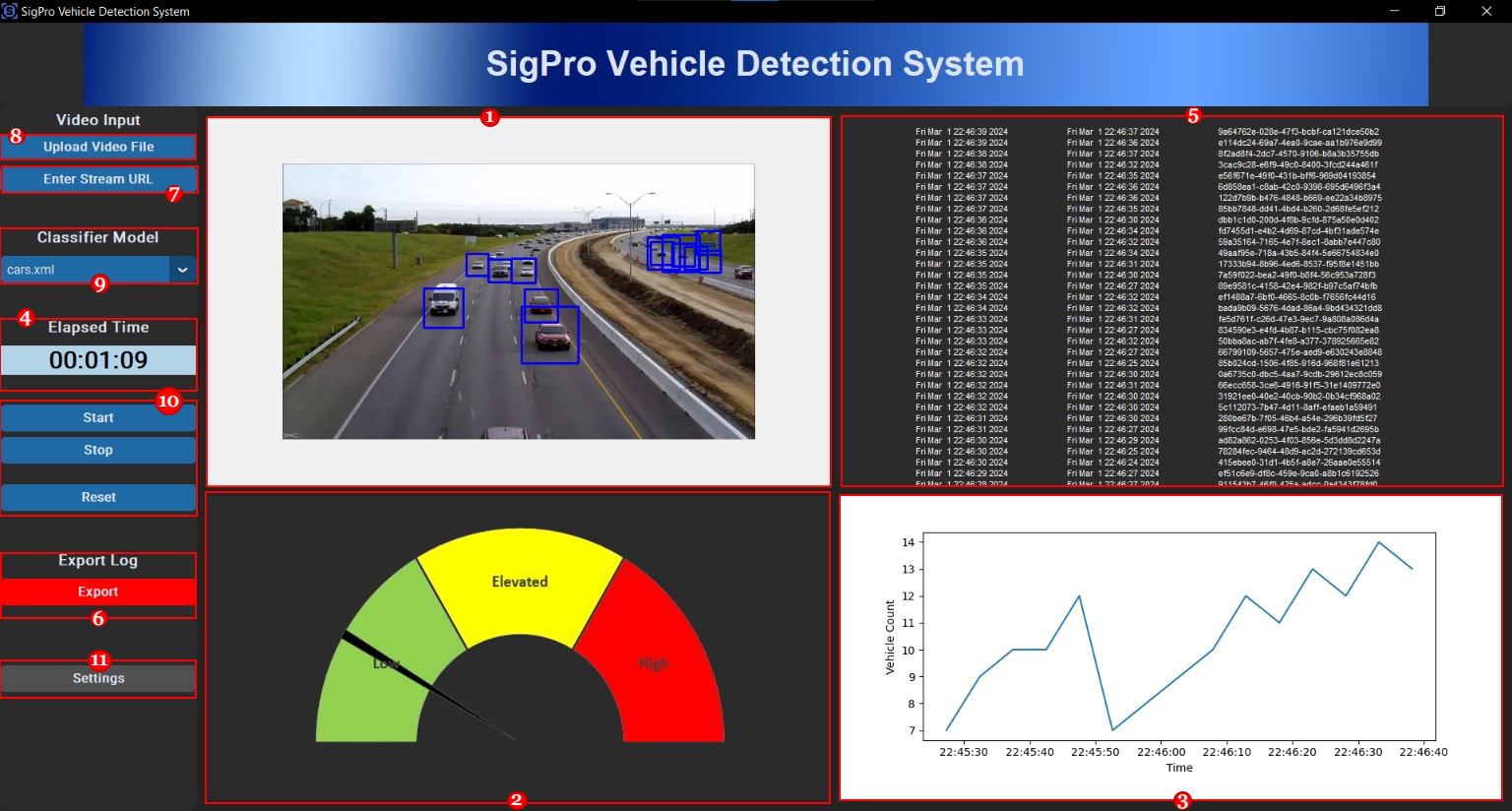
* CPU (Processor): Intel or AMD
* RAM (Memory): 16 Gigabyte or more
* Internet Connection

## Installation Instructions

To install the SVDA software, please follow the steps below:

1. Install Python 3.12 per vendor instructions.
   1. If using Windows, an environment variable needs to be added to use pip. Pip is located in the python installation directory in the Scripts folder.
   2. You can add the environment variable using the control panel or with the following command: setx PATH “%PATH%;path/to/pip”
2. Extract the contents of SVDA.zip.
3. Ensure the Computer has an internet connection.
4. Open a terminal and navigate to the SVDA folder. Run the following command: **pip install -r requirements.txt**
   1. If using Windows you can alternatively run the script SVDA\_Setup\_Windows.bat. This will install all the required dependencies.
5. To run the application, in the terminal run the following command **Python3.12 SVDA.py** (For Windows, **py SVDA.py).**
   1. If using Windows, you can alternatively run the script SVDA\_Run\_Windows.bat

## User Interface Overview



The SVDA user interface is designed to display selected videos, display data visualization & event logs, and provide file input/output functionality to the user. The screenshot above details the components in the main application user interface, while below lists each component’s descriptions.

Components

1. Main video window where selected traffic video will be displayed.
2. Traffic density gauge ranging from ‘Low Traffic’ to ‘High Traffic’
3. Line graph displaying Vehicle Count over time.
4. Elapsed time for a given video.
5. Log information for each vehicle detected.
6. ‘Export Log’ button.
7. URL selection field for specifying the URL of live traffic feed.
8. Upload Video File button for selecting a video file.
9. Classifier Model dropdown
10. Start, Stop, & Reset panel.
11. ‘Settings’ button.

## Using SVDA

Follow the steps below to take full advantage of this application.

***Process 1.*** Upload Video File

1. Open the application and select the ‘*Upload Video File’ (8).*
2. A dialog box should open for file selection. Select the desired file and hit the enter key.
3. If the desired video file is valid, it should start playing in the video display window *(1).*

***Process 2.*** Using Streaming Feed

1. Open the application and select the ‘*Enter Stream URL’ (7).*
2. A dialog box should open to provide the URL. Enter the URL and click OK.
   1. Example Stream URL: https://strmr3.sha.maryland.gov/rtplive/30001ac501a600d60057fa36c4235c0a/chunklist\_w943932104.m3u8
3. If the desired video file is valid, it should start playing in the video display window *(1).*

***Process 3.*** Download Log Files

1. Select the ‘Download Log File’ button *(6)*
2. A dialog box should open to specify a file name. Enter a file name and click Ok.
3. The log file will be saved in the application directory. The log file is a comma-separated value with ‘event id’, ‘*initial detection time’* (vehicle gets into the frame), and ‘*final detection time*’ (vehicle gets out of the frame)

***Process 4.*** Selecting Other Classifier

1. Select the ‘Classifier Model’ dropdown list
2. Select the desired classifier from the list.

***Process 5.*** Changing Settings

1. Select the ‘*Settings’* button *(11)*
2. A settings dialog will open. The following settings can be changed.
   1. **Graph Refresh Rate(Default=5000):** This is the Update rate of the graph and the gauge in milliseconds.
   2. **Elevated Traffic Threshold(Default=15):** This is the number of vehicles at which traffic will be classified as “Elevated”. Count below this level will be classified as “Low”
   3. **High Traffic Threshold(Default=25):** This is the number of vehicles at which traffic will be classified as “High”. Count below this level will be classified as “Elevated”
   4. **Movement Threshold(Default=0):** This is the number of pixels an object needs to move to not be classified as Stale. Default is 0, meaning any movement is classified as not stale.
   5. **Overlap Threshold(Default=0.6):** This is the percent of overlap between two detection boxes. Overlap greater than this value will be classified as the same object.
   6. **Stale Track Timeout(Default=5):** This is the time with out movement when the track will be dropped.
   7. **Scale Factor(Default=1.1):** How much the image is scaled for detection.
   8. **Minimum Neighbors(Default=2):** Number of neighbors each detection should have to retain it.

## Frequently Asked Questions & Troubleshooting

Please check the frequently asked questions if you run into a problem. Some of the questions you have may have been addressed below.

1. ***Which video formats are compatible with this software?*** The current version only works with MP4 and AVI formats for video uploads; and M3U8 formats for streaming.
2. ***Does the program break down the traffic density by lane?*** The current version only analyzes the average traffic of all lanes.
3. ***Can I import the data for the data visualization in the app?*** The data visualization components use the same data as is logged in the log file. You are free to download the CSV log file.
4. ***Do I need to have Python installed on my computer to run the software?*** Yes, follow vendor instructions to install python according to your system.
5. ***I get an error “py is not recognized as an internal or external command” or “python is not recognized as an internal or external command” or “pip is not recognized as an internal or external command”.*** This mean python is not installed on you machine. Install python 3.12.1 per vendor instructions. If you are using Windows and python is installed, verify the path to python.exe and pip.exe has been added to the system environment variables.
6. ***I get a “ModuleNotFoundError: No module named xxxxxx” error.*** This means the dependencies have not been installed. Run SVDA\_Setup\_Windows to install the dependencies. Alternatively, using a terminal, cd into the SVDA directory and run “pip install -r requirements.txt

## Additional Support

If frequently asked questions or troubleshooting has not addressed your questions and you need further assistance, please contact our support team (listed at the beginning of the page.)

# Test Plan (& Results)

## Introduction

* + ***1.1 Purpose.*** The purpose of the test specification document for the SigPro Vehicle Detection Application (SVDA) is to:
    - Outline the procedures used to test the list of functionalities of SVDA.
    - Detail required test environment.
    - Assign and communicate schedule and responsibilities to each task.
  + ***1.2 Scope.*** This test plan covers full system tests of SVDA. This includes:
    - User interface module
    - Functionality of each program module
    - Interface with other services
  + ***1.3 References***
    - Norman, S (2024). Project Plan for Vehicle Detection Application [Unpublished Assignment submitted for CMSC 495], University of Maryland Global Campus
    - IEEE Standard for Software Test Documentation (1998). IEEE Std 829-1998, 1-59, <https://ieeexplore.ieee.org/servlet/opac?punumber=5976>
  + ***1.4. Testing Environment***
    - 1.4.1 Test Hardware
      * A PC or a laptop that is running Windows, MacOS or Linux
      * Intel or AMD CPU
      * 16 Gigabytes of memory or more
      * Internet connection
    - 1.4.2 Test Software
      * A Windows, MacOS, or Linux operating system
      * Python 3.12.1 or later with the modules listed in section: Development Software
      * SVDA program
    - 1.4.3 Test Data
      * Video files in MP4 and AVI format
      * Access to M3U8 road traffic streaming services

## Test Items

* + ***2.1 User Interface Module.*** The user interface module allows the user to interact with the system. This module comprises an input (buttons, slide bars, etc.) and an output (vehicle traffic video display, traffic data visualization)
  + ***2.2 Main Program Logic.*** The main program logic is responsible for implementing the selected algorithm to identify and keep track of each vehicle in the street and analyze the traffic load.
  + ***2.3 Streaming Service Interface Module.*** The streaming service interface module allows the user to interface with 3rd party road traffic streaming services to analyze live traffic patterns.
  + ***2.4 Exception Handling Module.*** Any exception is handled by this module which allows the program to provide user relevant information and exit gracefully if necessary.

## Test Case Specifications

* + ***3.1 Black Box Test Cases***
    - 3.1.1 BBTC000 - User License Agreement
      * Simulates a user opening and accepting a license agreement
      * Prerequisite: Software file(s) downloaded to computer

|  |  |
| --- | --- |
| Input | Desired Output |
| 1. Launch Python file  2. Select “OK” | 1. License Agreement opens  2. Video loads in UI Widget |

* + - 3.1.2 BBTC001 - Test the File Upload UI
      * Simulates a user selecting/uploading a video file for analysis through the user interface
      * Prerequisite: User opens the application, accepts the license agreement and loads the main window with the file upload button visible (Specification ID: BBTC000).

|  |  |
| --- | --- |
| Input | Desired Output |
| 1. Click “Upload Video File” OR “Enter Stream URL”  2. Navigate to video file location, select file and press “Enter” OR Paste M3U8 link & press “Enter” | 1. File selection window opens, or URL Entry box opens  2. Video loads in Video Frame |

* + - BBTC002 - Test Video Player, Start and Stop UI Functions
      * Simulates a user playing an uploaded video through the user interface
      * Prerequisite: Video file is uploaded in the application (Specification ID: BBTC001)

|  |  |
| --- | --- |
| Input | Desired Output |
| 1. Click “Play” video player button  2. Click “Stop” video player button  3. Click “Reset” button | 1. Video plays, time increments  2. Video stops playing, time stops  3 Video, log, gauge, and graph panels reset to default |

* + - BBTC003 - Test Classifier Model Dropdown Menu
      * Simulates a user selecting a classifier model (xml) from available selectable classifiers
      * Prerequisite: Video file is uploaded in the application (Specification ID: BBTC001)

|  |  |
| --- | --- |
| Input | Desired Output |
| 1. Click the ‘Classifier’ dropdown  2. Select desired Classifier | 1. Dropdown expands, listing all available classifiers  2. Selected classifier displays in the “Classifier Model” Section |

* + - BBTC004 - Test Vehicle Detection Capabilities for Selected Video
      * Simulate a user running the program to detect a vehicle in a selected video
      * Prerequisite: Video file is uploaded, and classifier is selected in the application (Specification ID: BBTC003)

|  |  |
| --- | --- |
| Input | Desired Output |
| 1. Click ‘Detect Vehicle’ button  2. Click ‘Stop’ button | 1. A rectangle should be drawn around the vehicle detected, and should track the vehicle across the video frame as it moves  2. Video detection and tracking stops |

* + - BBTC005 - Test the application displays graph & gauge of vehicle traffic in the data visualization section of the user interface.
      * Simulate a user running the program to detect a vehicle in a selected video
      * Prerequisite: Video file is uploaded, and classifier is selected in the application (Specification ID: BBTC003)

|  |  |
| --- | --- |
| Input | Desired Output |
| 1. Click ‘Detect Vehicle’ button  2. Click ‘Stop’ button | 1. Vehicle detection count graph is displayed in the data visualization section of the user interface. Gauge displays the traffic status.  2. Video detection and tracking stops |

* + - BBTC006 - Test the application logs each detected vehicle in csv file with event id, initial detection time, last detection time (vehicle out of focus)
      * Simulates a user extracting vehicle log data file from the program
      * Prerequisite: Video file is uploaded, classifier is selected, and detection is initiated in the application (Specification ID: BBTC005)

|  |  |
| --- | --- |
| Input | Desired Output |
| 1. Click the “Export” button in the export log section of the user interface  2. Enter the name of the file. | 1. Dialog box opens prompting user to select log file destination folder  2. Log file with the provided name is saved. |

* + - BBTC007 - Test the exception handling
      * Simulates a user supplying invalid file for video input
      * Prerequisite: User opens the application and wait until the main page with the file upload button is visible

|  |  |
| --- | --- |
| Input | Desired Output |
| 1. Click Upload File button on the user interface  2. Navigate to location where the video file is and select (invalid) file and press Enter Key | 1. File selection window opens  2. Error message display in dialog box indicating invalid file format |

* + - BBTC008 - Test Settings
      * Simulates a user supplying invalid file for video input
      * Prerequisite: User opens the application and wait until the main page with the settings icon visible

|  |  |
| --- | --- |
| Input | Desired Output |
| 1. Click the ‘Settings’ icon  2. Edit ‘refresh rate’ minute field  3. Set aspect ratio from dropdown | 1. Settings window opens  2. Gauge & graph refresh rates update  3. The change reflects on the screen and the setting. |

## 

## Test Report & Traceability Matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Category** | **Description** | **Requirement Number(s)** | **Test Case ID** | **Pass / Fail** | **Results Data** |
| User Interface | User License Agreement | 1 | BBTC000 | Pass | Figure 1 |
| Upload/Download | File Upload, Stream Entry | 2, 3, 4 | BBTC001 | Pass | Figure 2, Figure 3 |
| User Interface | Start, Stop, & Reset Panel | 6 | BBTC002 | Pass | Figure 4 |
| User Interface | Classifier Model Dropdown | 11 | BBTC003 | Pass | Figure 5 |
| Streaming Interface | Vehicle Detection | 7, 8, 9 | BBTC004 | Pass | Figure 6 |
| Streaming Interface | Graph & Gauge Visualization | 12 | BBTC005 | Pass | Figure 7 |
| Upload/Download | Log Output & Functionality | 10 | BBTC006 | Pass | Figure 8 |
| Program Logic | Exception Handling | 5 | BBTC007 | Pass | Figure 9 |
| Program Logic | Settings | 1 | BBTC008 | Pass | Figure 10 |

***Text

Description automatically generated***

Figure 1-BBTC000 – License Agreement

Graphical user interface, application

Description automatically generated

Figure 2 - BBTC001 - Video File Dialog

Graphical user interface

Description automatically generated

Figure 3- BBTC001- URL Stream Dialog

Graphical user interface

Description automatically generated

Figure 4 - BBTC002 - Start, Stop, Reset

Graphical user interface

Description automatically generated

Figure 5 - BBCT003 – Classifier Model Selection

A group of cars on a road

Description automatically generated with low confidence

Figure 6 - BBTC004 – Vehicles being detected and Tracked

A picture containing text, device

Description automatically generated

Figure 7 - BBTC005 – Gauge and Graph Showing Traffic Data

Table

Description automatically generated

Figure 8 - BBTC006 – CSV File Saved

Graphical user interface, application

Description automatically generated

Figure 9 - BBTC007 – Invalid Video Error Handled

Graphical user interface

Description automatically generated

Figure 10 - BBTC008 - Graph Refresh Rate Updated to 1 Second

# Design

## Graphical user interface Description automatically generatedUI Design

Figure 11 - Preliminary Design Above, Current Design Attributes Below

## Classes, Methods, Fields, & Interfaces

### **SVDA.py**

1. App
   1. App()→None
   2. start()→None
   3. stop()→None
   4. reset()→None
2. DetectionLog
   1. DetectionLog(self, id, start, end) →None
   2. get\_record() →[self.id, self.start, self.end ]
3. UISettings
   1. UISettings()→None
4. Settings
   1. Settings() →None
5. SettingsDialog
   1. SettingsDialog() → None
   2. int\_validator() →bool
   3. float-validator() → bool
   4. cancel() →None
   5. ok()→None
6. TitleFrame
   1. TitleFrame()→None
7. CommandFrame
   1. CommandFrame()→None
   2. open\_settings()→None
   3. update\_elapsed\_time() →None
   4. start\_program()→None
   5. reset\_program()→None
   6. select\_algorithm() → None
   7. get\_video\_file() →VIDEO\_PATH
   8. get\_url() →VIDEO\_PATH
   9. open\_video\_input\_dialog()→filedialog.askopenfilename(), dialog.get\_input()→string
   10. export\_log()→None
8. VideoFrame
   1. VideoFrame() →None
   2. onVehicleTrackDropped() →None
   3. start\_program()→None
   4. play\_video() → bool
   5. update\_frame() →None
   6. useNumberOfVehiclesTracked() →None
   7. reset\_video\_frame() →None
9. GraphFrame
   1. GraphFrame()→None
   2. load\_graph\_background()→None
   3. plot\_graph()→None
   4. update\_graph()→None
   5. start\_graph()→None
   6. stop\_graph()→None
   7. reset\_graph()→None
10. LogFrame
    1. LogFrame()→None
11. GaugeFrame
    1. GaugeFrame()→None
    2. update\_image()→None
    3. start\_gauge()→None
    4. reset\_gauge\_frame()→None

### **VehicleDetector.py**

1. DetectorSettings
   1. DetectorSettings()
2. VehicleDetector()
   1. VehicleDetector(settings:DetectorSettings,haarModel:string) →None
   2. Detect(frame:MatLike) →tuple[int,MatLike]
   3. isTracked(bbox:RECT) →bool
   4. updateTrackers(frame:MatLike) →None
   5. registerVehicleTrackDroppedCallback(callback:FunctionType) →None
   6. vehicleTrackDropped(track:VehicleTracker) →None
3. VehicleTracker
   1. VehicleTracker(settings:DetectorSettings,frame:MatLike,bbox:RECT) →None
   2. update(frame:MatLike) →bool
   3. isSameVehicle(other:RECT)→ bool

## Data Structures, Data File Structures, Inputs, Outputs

* + ***Data Structures***
    - Log data will be stored in Arrays (download prints to CSV)
    - Video upload/stream will be stored in Database/Table
  + ***Data File Structures.*** Log Download: CSV File Output
  + ***Input Formats***
    - Video Inputs: Upload - AVI, MP4, Stream - M3U8
    - Classifier(s): XML
  + ***Output Formats.*** Log - CSV, TXT

## Approach & Performance Estimates

***Approach.*** The first design priority will be building the classes for the video output. Whether it be configuring the different classifiers and training them, perfecting vehicle detection/identification & tracking their time on screen, enabling that video player to be live, or even making the data loggable. From there, we will shift clockwise - next to configuring the log printing & distributing output, then to transferring the data in graph form, then having that graphed data be analyzed and gauged, and finally ensuring that the distributed log file output is downloadable as a CSV file.

Next will be configuring and verifying actual uploads - whether using a file chooser for the manual feed or ensuring that stream links can extract the desired video file. The three last items to approach will be the elapsed time clock, the settings menu, and the GUI which includes the (although the window can also be configured first, despite being the last design priority).

***Performance Estimates.*** The performance should work in this way, following the design:

1. Show the actual video or the live stream of the road with cars.
2. Show the percentage of crowd on the road with cars.
3. The number of cars displayed and the traffic density per road.
4. Elapsed time of the video/live streaming.
5. Log of each car appearing and disappearing.
6. By clicking the “Download” button, the text file of the log is downloaded.
7. By entering the URL of the streaming, it will appear on the screen no.1.
8. By entering the PATH of the video, it will appear on the screen no.1.
9. Classifier selections will be shown with the drop button.

## [Policy of Software](https://docs.google.com/document/d/19ogwIAlI_EiycIJB7824n0FCliZQ-8FvXSnclqtkn3U/edit?usp=sharing)

* + Updated 2/2, 2:30 AM EST

# Development History

## Phase 1

* + ***Overview.***The goal for Phase 1 was to set up the structure of the project and create an application with a basic UI to load and display an image. This required researching and becoming familiar with OpenCV as well as integrating it into our project. Additionally, we had a goal to train a classifier for use in the project. This involved collecting hundreds of positive and negative images and running a training algorithm on them.
  + Status: On Schedule

All Phase 1 Milestones are Complete.

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase 1** | **Due Date** | **Responsible** | **Status** |
| Project Structure Setup | 2/7/2024 | TJ, Tomoki | Complete |
| Initial App that loads & displays an image | 2/10/2024 | TJ, Tomoki | Complete |
| Basic UI created | 2/10/2024 | Senay, Freddy | Complete |
| Train Haar Classifier | 2/12/2024 | Stephen | Complete |

* ***Problems Encountered.*** No major problems were encountered during this phase.
* ***Reevaluation.*** Initially, our method of object detection was going to use a Haar Cascade classifier. After looking over the OpenCV documentation, another method became apparent. This method involves using background subtraction to detect individual silhouettes. A decision on which method to use will be made as the project progresses. Additionally, since the OpenCV API is similar to displaying images and video, we decided to load and display video instead of images.

Figure 12 - Phase 1 Results

## Phase 2

* + ***Overview.*** During the second phase of our project, our main goal was to create an initial draft of the user interface (UI) and implement vehicle detection within video streams. This involved researching Python UI libraries as well as exploring object detection techniques. For the UI, we chose tkinter. Tkinter is a python interface for the Tcl/Tk gui toolkit and supports Linux,Mac, and Windows. The detector is using a Haar cascade classifier using a open source pre-trained model that we found on GitHub.
  + Status: On Schedule

All Phase 2 Milestones Are Complete.

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase II** | **Date** | **Responsible** | **Status** |
| Vehicles detected in image | 2/15/2024 | TJ, Tomoki | Complete |
| App loads and displays video from file | 2/17/2024 | TJ, Tomoki | Complete |
| App loads and displays video from stream | 2/18/2024 | TJ, Tomoki | Complete |
| First Draft of UI complete | 2/17/2024 | Senay, Freddy | Complete |

* ***Problems Encountered.*** A couple of minor issues were encountered during this phase.
  + Detection time was taking too long. The video needed to be resized smaller to reduce the detection time.
  + The Panel library that we are using was meant for web app development. A workaround was created to view the gauge in the browser. We’re researching building the library from scratch so it will work natively, but this may not be feasible.
* ***Reevaluation.*** Initially, our method of object detection was going to use a Haar Cascade classifier. After looking over the OpenCV documentation, another method became apparent. This method involves using background subtraction to detect individual silhouettes. We are still evaluating these two methods.

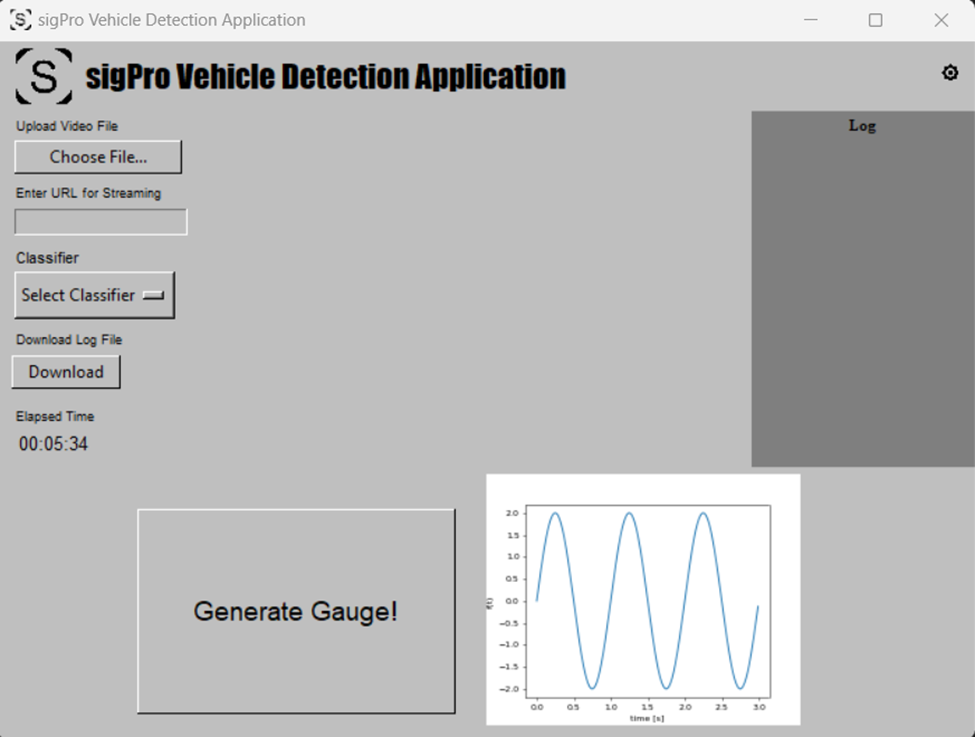


Figure 13 - Phase 2 Results

## Phase 3

* + ***Overview***. During the third phase of our project, our main goal was to have nearly all the functionality complete as well as the UI complete. The UI was redesigned to make the application more polished and user-friendly. Also, our application is now able to detect and track vehicles in video files as well as streams.
  + Current Status: On Schedule

All Phase 3 milestones are complete.

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase III** | **Date** | **Responsible** | **Status** |
| App Detects vehicles in video | 2/22/2024 | TJ, Tomoki | Complete |
| App logs detections information | 2/25/2024 | TJ, Tomoki | Complete |
| Compare our classifier with an open-source classifier and choose the best one for the application. | 2/25/2024 | Stephen | Complete |
| Main UI Complete | 2/24/2024 | Senay, Freddy | Complete |

* ***Problems Encountered.*** Detecting and tracking objects in video is a resource-heavy task. Performing this on real-time video could lead to video issues. During preliminary testing, processing a frame can take anywhere between 20ms to over 150ms depending on the number of objects being tracked and the resolution of the video. We are investigating methods to speed up detection.
* ***Reevaluation.*** At this point in time, everything in Phase 3 is going to plan. Cross communication, suggestions, and improvements alike streamline and redefine this part of the project – from classifier model shifts, to the integration of application themes, to the refining of graph & gauge data metrics.

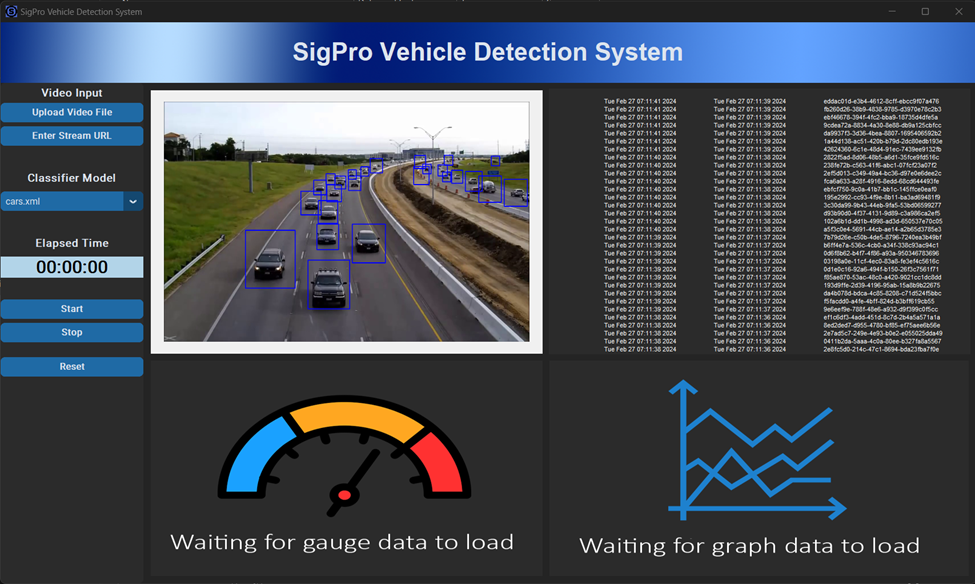


Figure 14 - Phase 3 Results

# Conclusion

The entire process of building this program - from the documentation stages to each of the three phases, all shaped the final product in overt and subtle ways. While aspects of the project such as the specifications, rationale, and the phase items remained as they were in the final product; other attributes such as the design and overall programming went through many different iterations. Overall, the Vehicle Detection Application accomplishes what it’s intended to - intaking two separate video formats, outputting a log and two methods of data visualization, and making the log available for download. However, the journey to that final product was not a linear one.

While this journey spurred many lessons, there were three that were most crucial. The first was that both constant synchronous *and* asynchronous communication were key to the project’s success. While delegating tasks (as seen in the project plan) that were fairly autonomous utilized asynchronous methods such as Slack messaging or email better, we found that for clarifications’ sake or progress checks that synchronous communication (i.e. Huddles in Slack) worked best. We also learned to remain flexible - both in terms of things within our control and unforeseen circumstances. Lastly, a lesson our group learned more in practice was to be aware of our resources. This resourcefulness ranged from applying previous classes’ principles, to digital aid (IDE help guides, YouTube, programming forums, etc.), to reliance on each other.

Those lessons were the catalyst for a nuanced, honest review of our process - the good *and* the bad. In hindsight, some of the strengths of the design included the comprehensiveness of its initial documentation, the linear flow of the program’s logic from launch to log data downloading, and the simplicity of the interface. There were also areas that the group could do better, such as investing more time on additional features (i.e. more settings and log export formats) or implementing fail-safes & backups. In terms of future improvements for the program, addressing those design limitations earlier within the process - which may involve implementing time to tinker with prototypes beyond requirement functionality at each of the phases - will make future projects less prone to those design limits. Additionally, when it comes to improving the project management itself - developing an agenda for each meeting with actionable items for members would cut down on uncertainty and allow for more productive check ins.

In conclusion, while the process for the program’s development is complete and objectives have been successfully reached, this project is by no means the end of the learning process. From a milestone achievement perspective (documentation, phases) to a retrospective one (strengths, weaknesses, lessons learned, etc.), this project not only allowed us to acquire practical computer science experience, but also gives us the opportunity to reflect on our preferred collaboration styles and affinities. Both of these are (and will be) useful to members beyond the scope of this course.