Road Mapper

By

Muaaz Deyab

Mohammed Zeidan

Advisor

Mr. Antoine Aouad

Senior Project Submitted to the Faculty of Arts and Sciences

Department of Computer Science

AMERICAN UNIVERSITY OF SCIENCE & TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in

Computer Science

Achrafieh

04/06/2021

Road Mapper

By

Muaaz Deyab

Mohammed Zeidan

Advisor

Mr. Antoine Aouad

Senior Project Submitted to the Faculty of Arts and Sciences

Department of Computer Science

AMERICAN UNIVERSITY OF SCIENCE & TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in

Computer Science

Achrafieh

04/06/2021

We certify that we have read and tested this project and that, in our opinion, it is satisfactory in scope and quality as a senior year project for the degree of Bachelor of Science in Computer Science.

Project Committee

Elie Nasr, Ph.D.,

Antoine Aouad, M.S., MBA,

Charbel Boustany, M.S.,

Acknowledgments

We wish to convey our sincere gratitude and appreciation to each and every person who helped us through this project.

Special thanks are due to Mr. Antoine Aouad for his help and assistance as well as his for his guidance throughout the entire project.

Abstract

This paper presents an approach to improve Autonomous Vehicles in order to have a better navigation during tough weather conditions or random road situations like vision blockage by other vehicles on the road that makes it difficult for the recognition cameras or radars to identify traffic signs in real time.

The fact that in most countries traffic signs are not something that changes too often allows this project to present the idea of a system that uses the same complicated Artificial Intelligence algorithms (OpenCV, TensorFlow, Keras) to recognize the type of traffic signs in environmentally controlled conditions on a specified road. After that it takes their location, type and all needed information and stores this data in a table and adds this table to the navigation system of the vehicle to aid it with accurate, stable measurements.

This project aims to amalgamate the aspects of Real Time Recognition and Street View Map Navigation Systems in Self-Driving cars or Autonomous Vehicles that are using real time recognition to identify the traffic signs by adding a system that takes the location of these signs and then checks it on a map to help assure that the decision made by the vehicle is more trustworthy in all road situations and weather conditions.

Autonomous Vehicles research indicates that the problem with the navigation system of many vehicles is that the decision based on using the data provided by real-time recognition systems to identify traffic signs is not trustworthy in inconvenient conditions such as curved highways, however real-time readings have another variable to consider the trustworthiness of; that variable is the accuracy level of the algorithm used to detect the traffic signs in the images/videos provided, which depends on many factors for example, how close the vehicle is to the sign. So the entropy of the vehicle on the road is high which in turn could lead to tragic car accidents.

To summarize, this project aims to eliminate the variables of reading real-time data from the road directly making it more efficient to use recognition systems while also providing a scalable, and trustworthy solution.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION AND BACKGROUND

1.1 – Autonomous Vehicles

Nowadays vehicles have evolved from their normal basic functions of transportation from point A to point B to being more complicated where you can do more than just sit behind the wheel in your vehicle without worrying about getting in a car accident.

1.2 – Traffic Signs Recognition Systems

Autonomous Vehicles are equipped with much more than what normal vehicles have. For example, they have a lot more sensors and cameras than normal vehicles which allows them to have a better perspective of the location of the vehicle as it moves on the road.

Autonomous Vehicles use many different AI algorithms to read traffic signs on the side of the road. These algorithms read the signs in different ways some of them recognize the text on the sign then its shape and some of them are models that were trained to recognize the picture of the sign.

1.3 – Need for the Project

The AI algorithms used to detect the signs have different factors to take into account such as the accuracy of the decision made or the fluctuation of the entropy as the conditions change every time the vehicle moves on the road and many other factors which results in making the decision made by the AI less reliable each time the vehicle has to detect a sign.

This Project aims to provide a solution to this problem by using environmentally controlled readings provided by google maps street view, however reading each sign on a every road is a very time and resource consuming process which is why we are using the Artificial Intelligence algorithms that are currently being used in Autonomous Vehicles in coherence with google street view API and map reading algorithms to make a unified platform that helps to ensure that the Autonomous Vehicle gets provided with a more reliable, consistent and trustworthy decision.

1.4 – Conclusion

To conclude, this project uses Artificial Intelligence algorithms to identify the signs on a specified road and then it automatically takes their location as well as their type and adds it to the navigation system of the Autonomous Vehicle in a table format.

CHAPTER 2

PROBLEM STATEMENT

2.1 – Technical Difficulty

As Autonomous Vehicles get on the road they are naturally faced with many different challenges and some of these challenges are unsolvable which is why Autonomous Vehicles are not fully autonomous but rather more assistive.

With that being said, few of the challenges are vision blockage; when a vehicle is on a heavy traffic road or on a turning highway it is hard for the sensors, radars, and the cameras to detect the exact position of the vehicle due to either being blocked by other vehicles or in the case of turning roads, sensors are not being able to properly measure the distance far enough in order to for the vehicle to stop suddenly in case of emergency.

2.2 – Traffic Signs Recognition Problems

Sign Recognition Systems have always faced challenges with weather inconsistency. In some countries the weather is constantly snowing or raining and Autonomous Vehicles have to drive with everything covered in snow which causes a huge problem for Recognition Systems because some approaches to detect signs use Convolutional Neural Network (CNN) to extract the colors of the signs followed by Support Vector Machine (SVM) to classify the sign. With this approach snowing weather affects the color detection part negatively causing a decrease in accuracy of detecting signs.

A different approach uses Histogram Oriented Gradients (HOG) and Local Binary Pattern for the extraction of the shape of the signs then detection, after that it inputs the results into a Machine Neural Network (MNN) to classify the signs. With this approach a problem immerges where the vehicle cannot properly detect the sign shape in long distance roads that have high curvature.

2.3 – Conclusion

To conclude, Autonomous Vehicles face many challenges when detecting signs on the side of the road some of these challenges are caused by a hardware limit and some of them are limited on the software side.

CHAPTER 3

EXISTING SOLUTIONS

3.1 – Similar Companies

Autonomous Vehicles have been adopted by many companies because it became one of the necessities today.

3.1.1 – WAYMO

Is considered the top company in this field and its vehicles have registered enormous number of kilometers driven (around 32 million kilometers in real life and more than 16 billion in simulation).[1] They are leading because they're using high-end technologies in their vehicles, in addition to the cameras, radars, and sensors. They also use microphones to detect sirens from emergency vehicles for autonomous functionalities and currently they are working on fully driverless vehicles.

3.1.2 – Cruise Division

Has the world's second autonomous fleet with over 180 vehicles under testing where they have driven more than 1.6 million kilometers.[1] Its partnership with huge companies like General Motors and Honda allows it to be able to produce huge number of vehicles where its partners provide the vehicle design and designs AV software and hardware.

3.1.3 – Tesla

Is one of the biggest companies in the world, it has more than 600,000 vehicles on global roads. These vehicles are provided with eight cameras surrounding the car to provide 360 degrees of visibility up to 250 meters and with ultrasonic sensors, radar with enhanced processing which makes a complete system that can detect objects in any situation. It is expected to have a million of fully driverless vehicles on the roads in 2021[1].

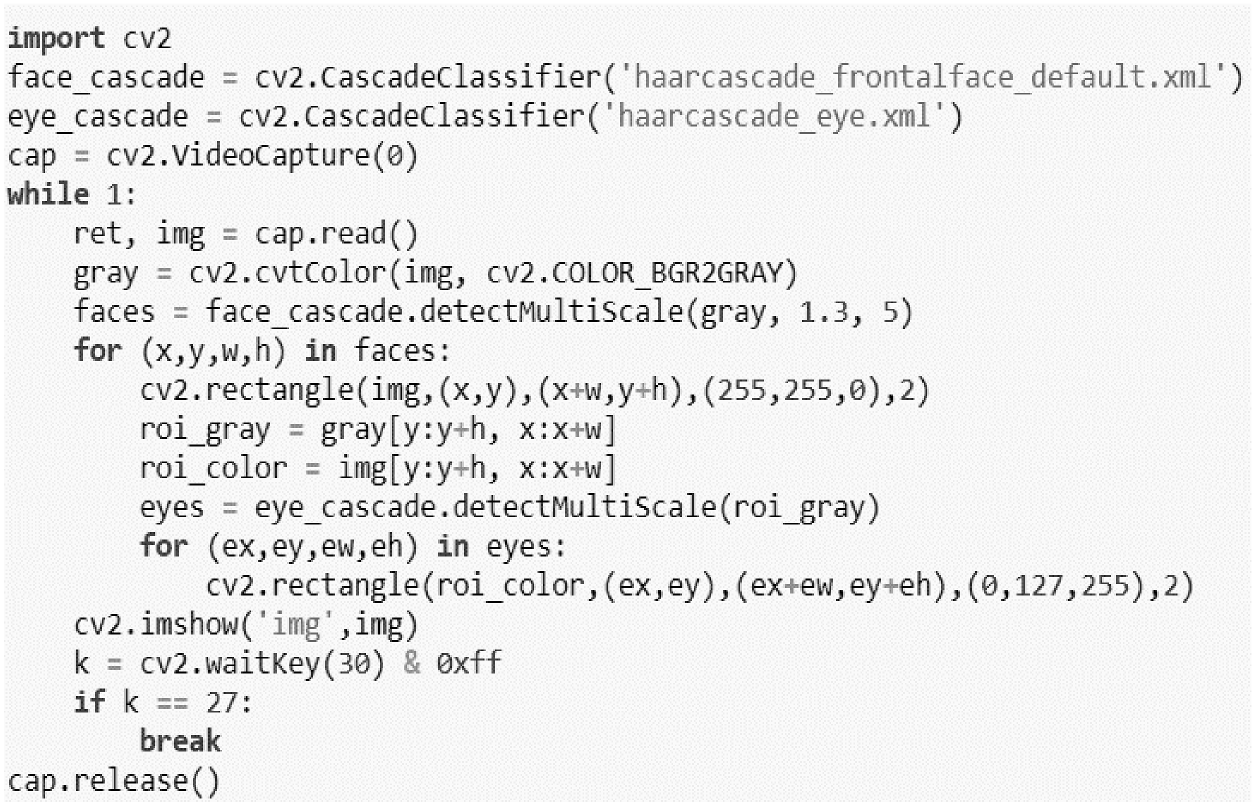
3.2 – Used Algorithms

Many algorithms are used in order to build a fully functional object recognition system such as OpenCV, TensorFlow, CV2, PYTORCH, and many other algorithms that have proven to provide us with acceptable results in the right circumstances.

3.2.1 – OPENCV

OpenCV is an open-source computer vision and image processing library. It was developed by Intel in the early 1999 and was officially released in 2006. It includes thousands of computer vision algorithms, and is developed primarily in C++ and wrapped by Python, Java, and JavaScript. It can run on all platforms like WINDOWS, LINUX, and MAC OS. It has huge variety of features like image read and write, video capturing and manipulation, objects detection. In this project, we will use this library for traffic sign detection and video capturing.

OpenCV will take a video capture and then cut this video into frames and recognize each frame to apply the image processing techniques on it, where a trained model will be used to decide what to do. Also, we will use OpenCV to determine the distance between the vehicle and the sign. In order to determine the distance, we are going to use triangle similarity which is D = (W x F)/P where W is the width of the object which can be easily recognized after drawing a rectangle around the object, F is the focal length of the camera and P is the pixels covered by the camera. Figure 1.1 is a sample of how to detect faces using OpenCV.[2]



```
import cv2
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
cap = cv2.VideoCapture(0)
while 1:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,255,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]
        eyes = eye_cascade.detectMultiScale(roi_gray)
        for (ex,ey,ew,eh) in eyes:
            cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,127,255),2)
    cv2.imshow('img',img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cap.release()
```

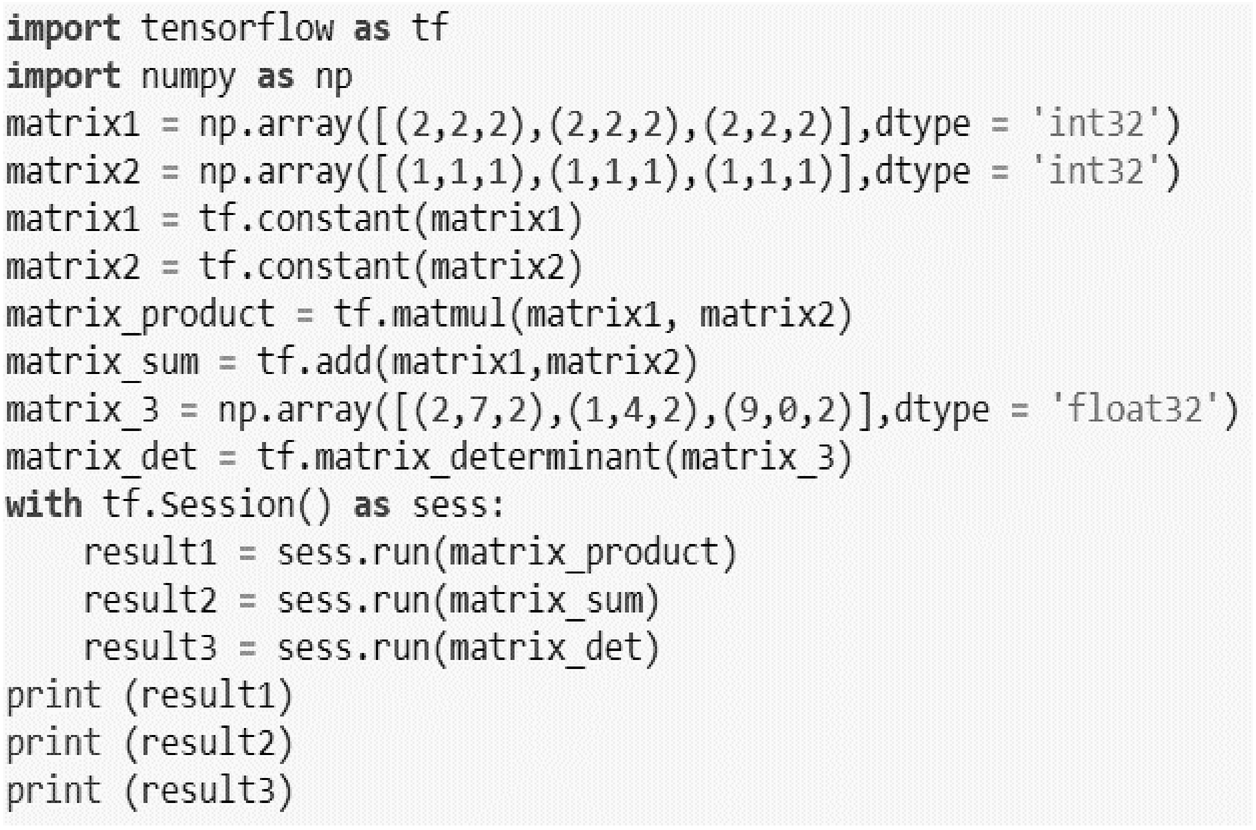
Figure 3.1 – OpenCV Sample

3.2.2 – TensorFlow

TensorFlow is an open-source Machine Learning framework used for complex mathematical computation and large-scale Machine Learning, and it is mostly used to create and train deep neural network.

It works by creating data graphs, structures or series of nodes which basically is a mathematical operation and each node is connected to other nodes by edges which represents a multi-dimensional array (Tensor), and there are many libraries for the same purpose like PYTORCH, and KERAS.

The code in figure 1.2 will build and run a basic Neural Network using TensorFlow.[3]



```
import tensorflow as tf
import numpy as np
matrix1 = np.array([(2,2,2),(2,2,2),(2,2,2)],dtype = 'int32')
matrix2 = np.array([(1,1,1),(1,1,1),(1,1,1)],dtype = 'int32')
matrix1 = tf.constant(matrix1)
matrix2 = tf.constant(matrix2)
matrix_product = tf.matmul(matrix1, matrix2)
matrix_sum = tf.add(matrix1,matrix2)
matrix_3 = np.array([(2,7,2),(1,4,2),(9,0,2)],dtype = 'float32')
matrix_det = tf.matrix_determinant(matrix_3)
with tf.Session() as sess:
    result1 = sess.run(matrix_product)
    result2 = sess.run(matrix_sum)
    result3 = sess.run(matrix_det)
print (result1)
print (result2)
print (result3)
```

Figure 3.2 – TensorFlow Sample

3.2.3 – SciKit-Learn

Just like TensorFlow, the SciKit-Learn library is an open-source machine learning library for supervised and unsupervised machine learning. It provides the programmer with lots of functionalities to help in preprocessing data such as Linear and Logistic Regression, Classification, and Clustering. This powerful library is great for these kinds of systems since it is built upon many technologies and libraries like Pandas, and Matplotlib.

3.3 – Conclusion

There are many existing solutions to the problem we have; however, none of them solve the real problem that a vehicle faces on the road because most of them focus on the technicalities of the issue while forgetting about the actual problems that the vehicles have.

CHAPTER 4

PROPOSED SOLUTION

4.1 – Model Training

We are going to use TensorFlow and neural networks to train our model. TensorFlow is one of the most popular libraries for machine and deep learning, which is why we chose it over other libraries like PYTORCH which provides more functionality however, TensorFlow is built using C++ which makes it much faster than PYTORCH. [4]

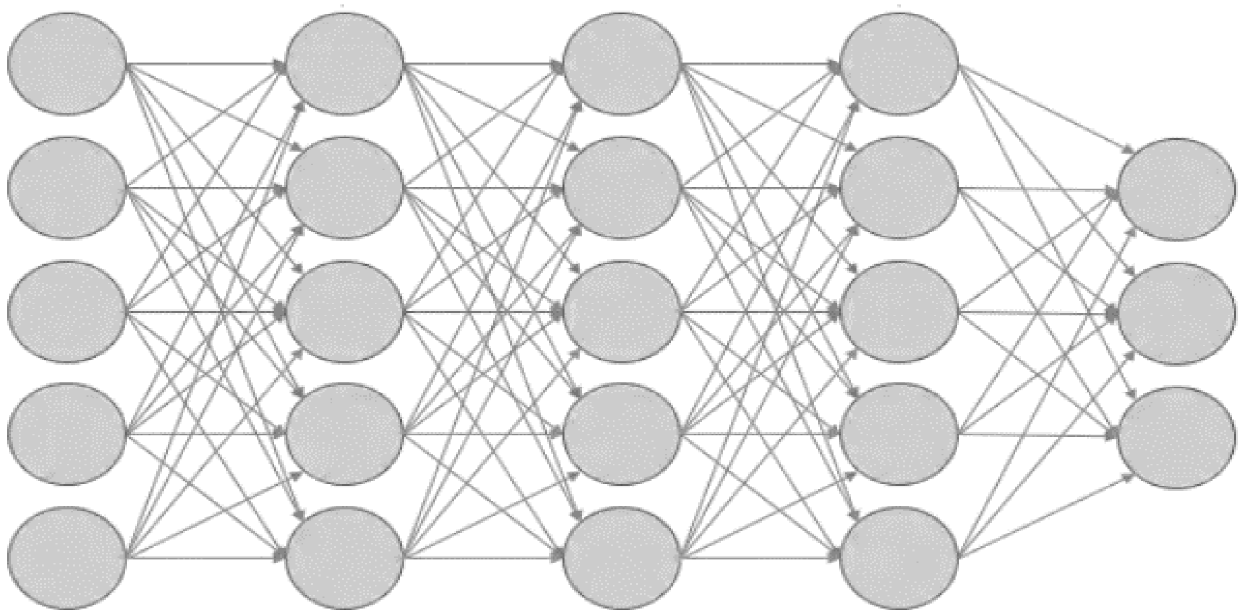


Figure 4.1 – Inter-connected Neurons

A neural network is a series of algorithms that recognize the underlying relationships in a set of data by simulating a human brain operation and constructing virtual neurons that interact with each other to finally end-up with a prediction. Neural networks consist of multiple layers of inter-connected neurons as shown in figure 1.3 [4] each neuron represents a mathematical function

4.2 – Sign Recognition

To put it simply, the Vehicle has a system that consists of a forward-facing camera, which scans the road ahead for traffic signs. This camera is connected to character recognition software, which then makes a decision based on the instructions provided by the sign.

To get into more details we can start by describing how the character recognition software works. As we have described in chapter 3, there are many algorithms that can be used to detect objects; however in our project, we chose to use OpenCV simply because it's the best. OpenCV is an open-source computer vision library that works in both C++ and Python, it is prebuilt with all the necessary techniques and algorithms to perform several image and video processing tasks. It’s quite easy to use and this makes it clearly the most popular computer vision Library. It also is multi-platform, allowing it to support applications for Linux, Windows and Android. [5]

4.3 – Maps Usage

Google Maps is a mapping service developed by Google which provides real-time location alongside with many other like satellite imagery, terrain view, and street-view. We will use street-view to take pictures from car perspective every 10 meters, street-view provides a 360 panoramic street-level view for almost all public streets Figure 4.2 represents a sample of street-view near The AUST University Achrafieh campus [2].

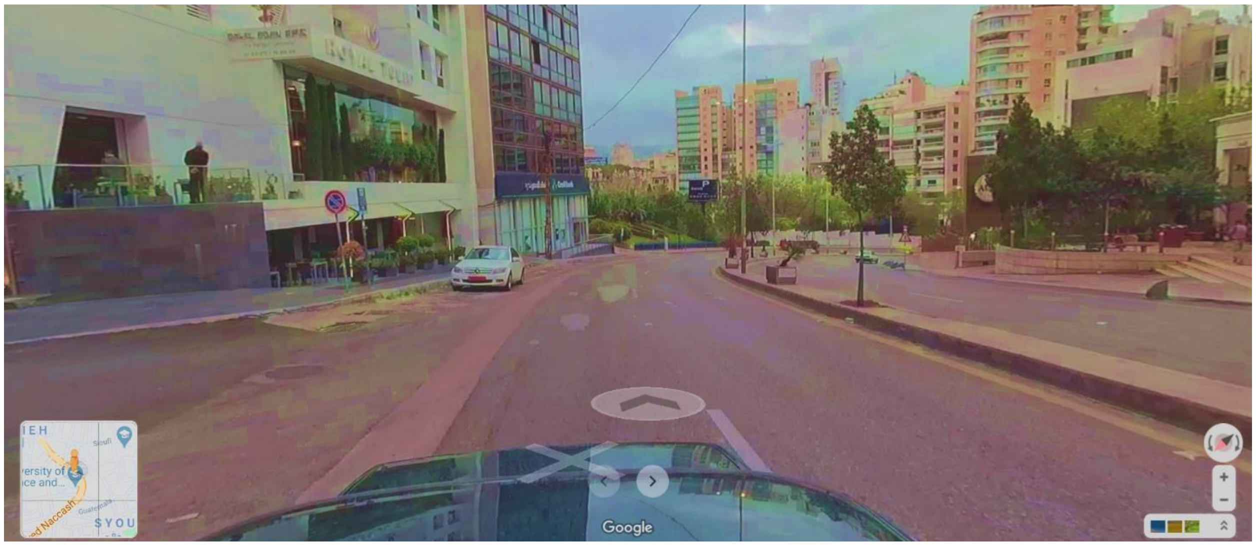


Figure 4.2 – Google Street View

4.4 – Conclusion

To conclude, for our model training stage we will be using an XML file generated from a TensorFlow trained model that has all the guidelines to detect almost all traffic signs from an image series brought from the google street-view API then for the detection stage we will input this xml file into OpenCV in order to actually detect the signs in every 10 meters on the road and in the final stage we will take all sign locations and provide them to the navigation system of the vehicle in a table format.

CHAPTER 5

DESIGN SPECIFICATIONS

5.1 – Context Diagram

A context diagram shows the different entities that the system is composed of. Context diagrams are used to determine what lays outsides the system boundaries. In the below context diagram six modules exist that shall be integrated together to form the main system which is “Road Mapper”. In other words this diagram represents the application’s core business logic.

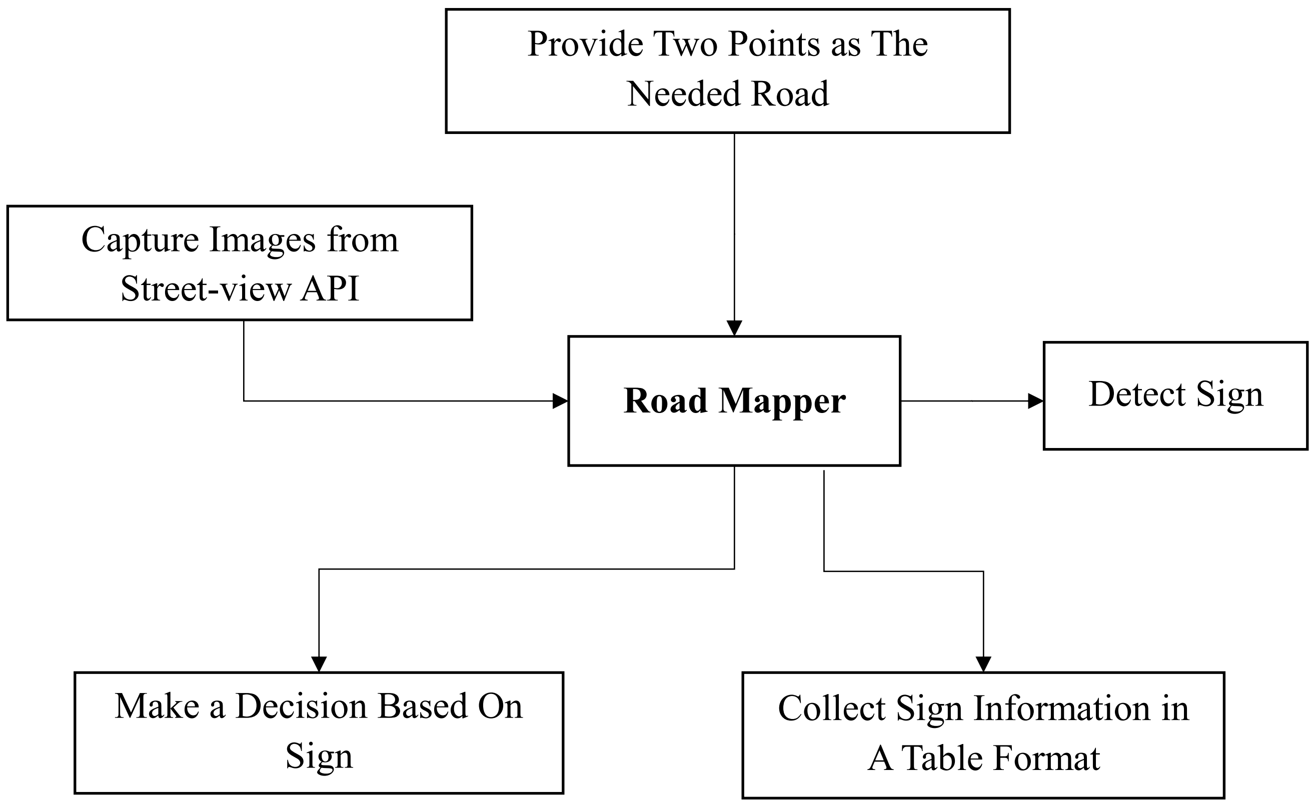

Provide Two Points as The Needed Road
Capture Images from Street-view API
Road Mapper
Detect Sign
Make a Decision Based On Sign
Collect Sign Information in A Table Format

Figure 5.1 - Context Diagram

5.2 – Use Case Diagram

A use case diagram is a set of possible sequences of interactions between systems and users in particular environment and related to a particular goal. It should contain all system activities that are significant to the actors. The below figure shows the system boundaries, along with the different activities that an actor can perform.

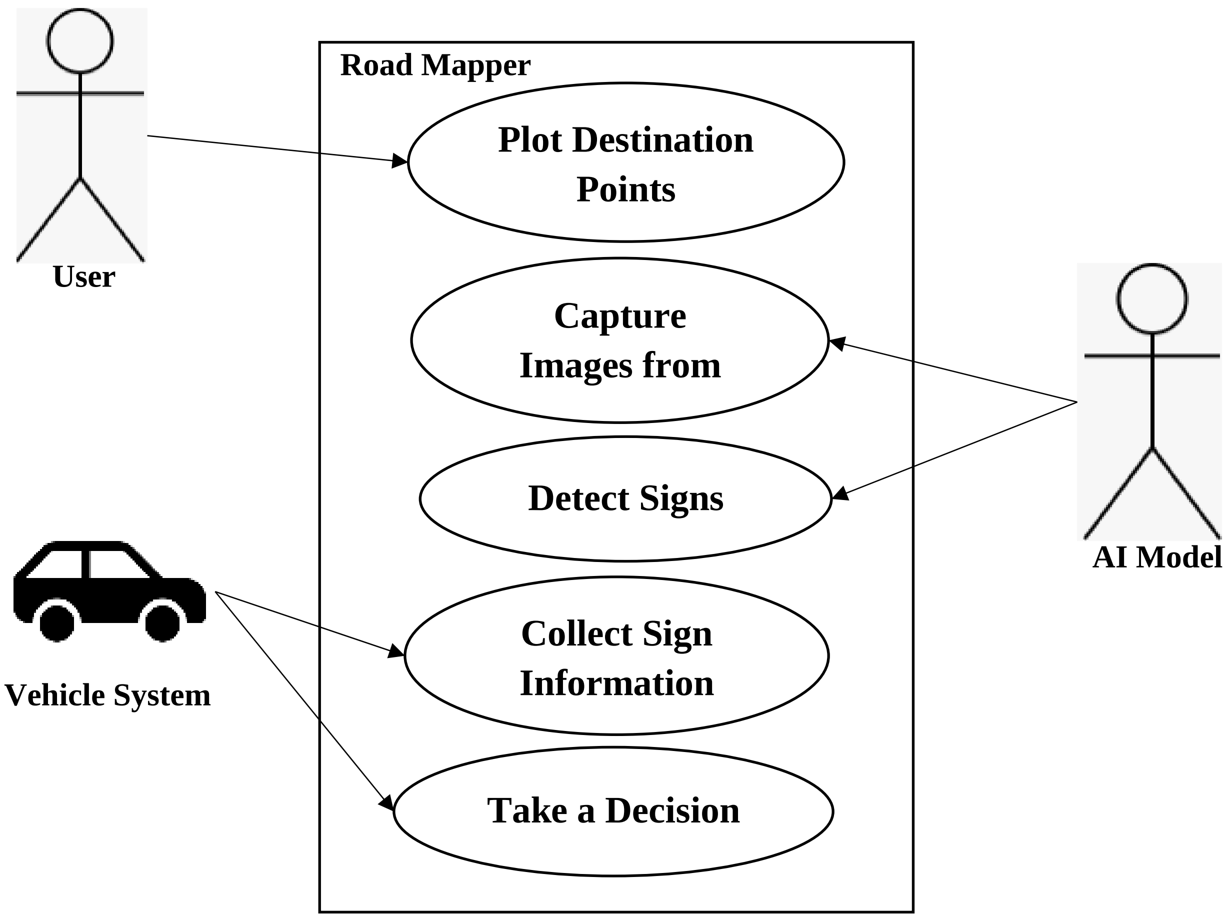


Figure 5.2 - Use Case Diagram

5.3 – Sequence Diagrams

A sequence diagram is the collaboration of objects based on time sequence. It captures the sequence of the interactions between user and the system or between different systems by showing the order of messages sent, the content of each message, and when they are sent. The below sequence diagrams show a general/basic overview of the entire application.

For destination and Location selection:

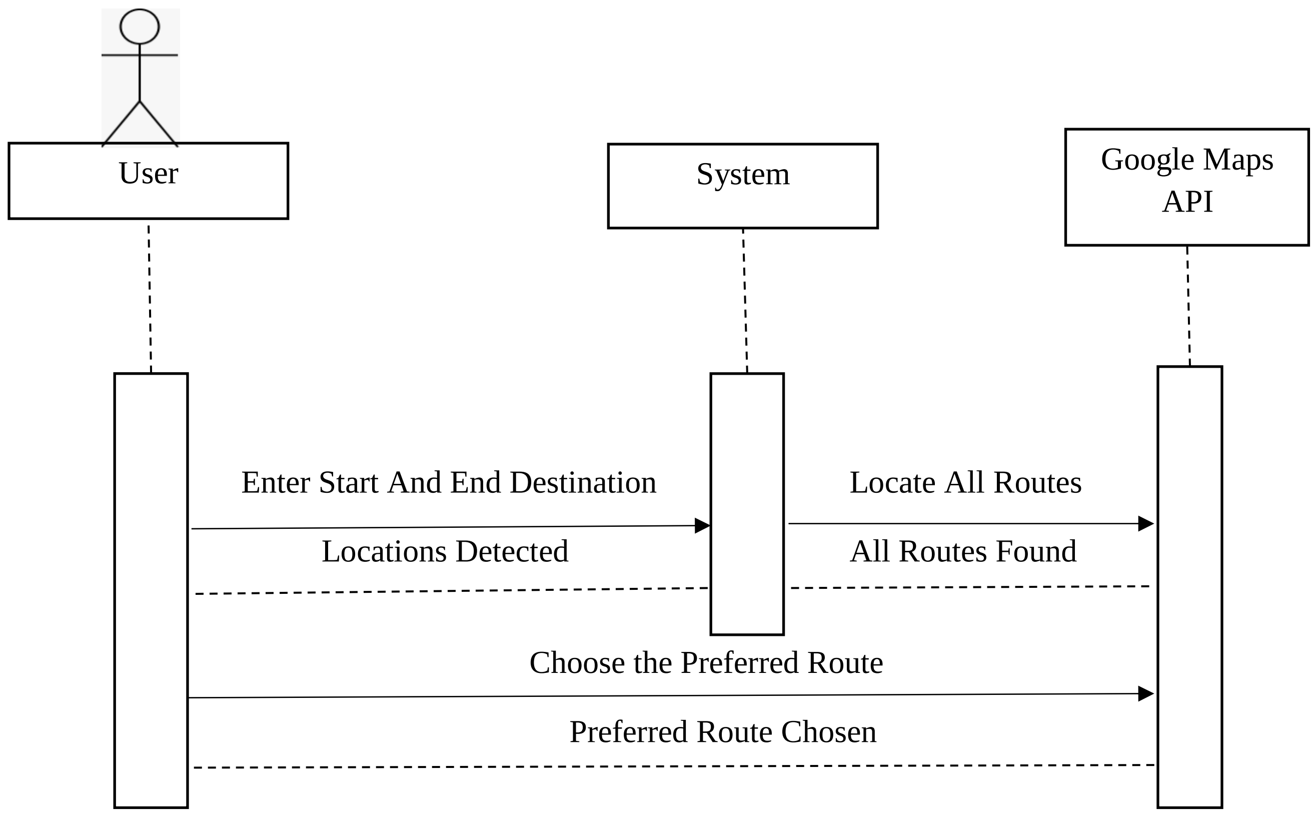


Figure 5.3 – Destination and Location Sequence

For scanning and detecting signs:

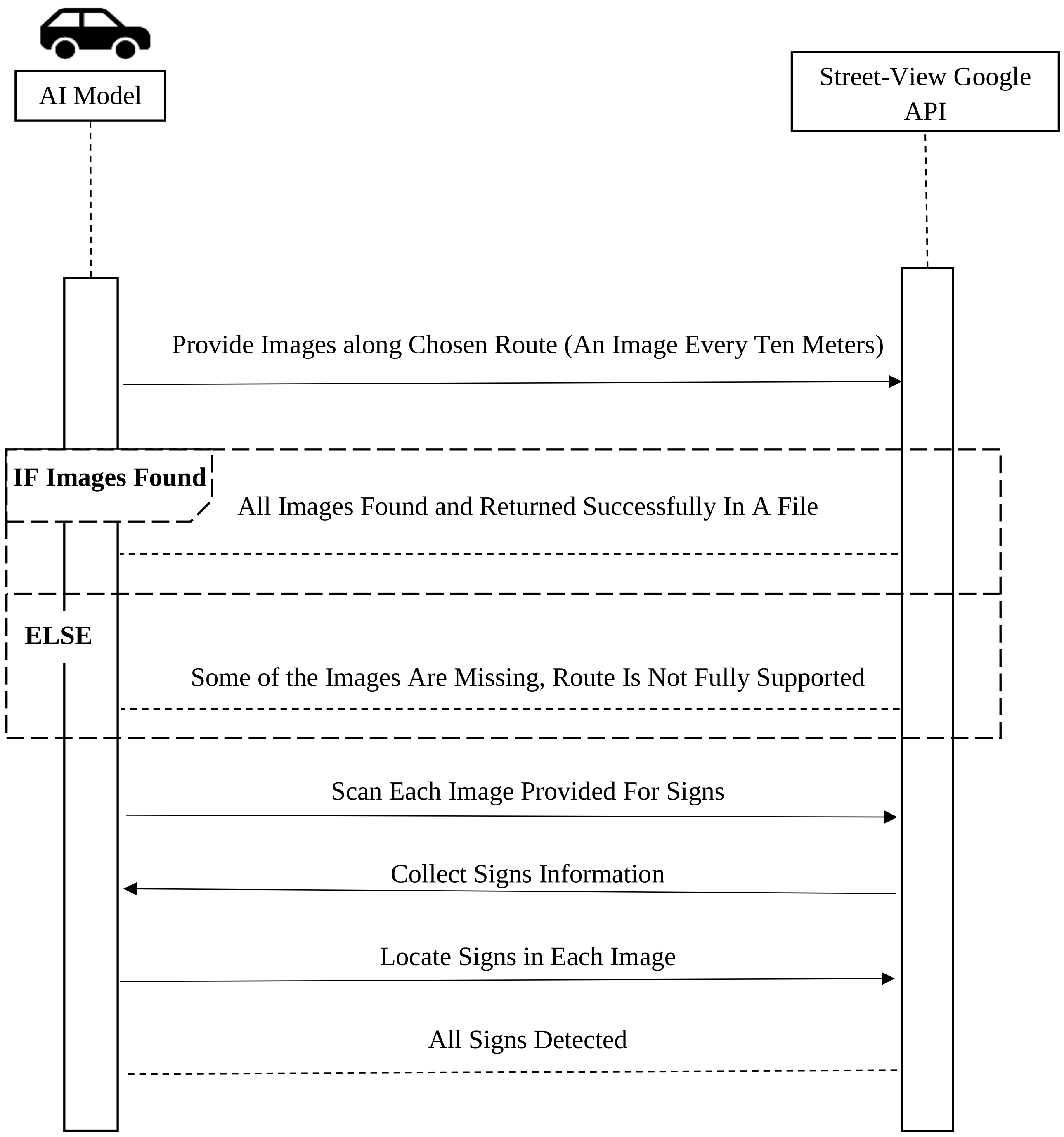

AI Model
Street-View Google API
Provide Images along Chosen Route (An Image Every Ten Meters)
IF Images Found
All Images Found and Returned Successfully In A File
ELSE
Some of the Images Are Missing, Route Is Not Fully Supported
Scan Each Image Provided For Signs
Collect Signs Information
Locate Signs in Each Image
All Signs Detected

Figure 5.4 – Sign Locating Sequence

For collection of sign locations to be provided to the navigation system of the vehicle:

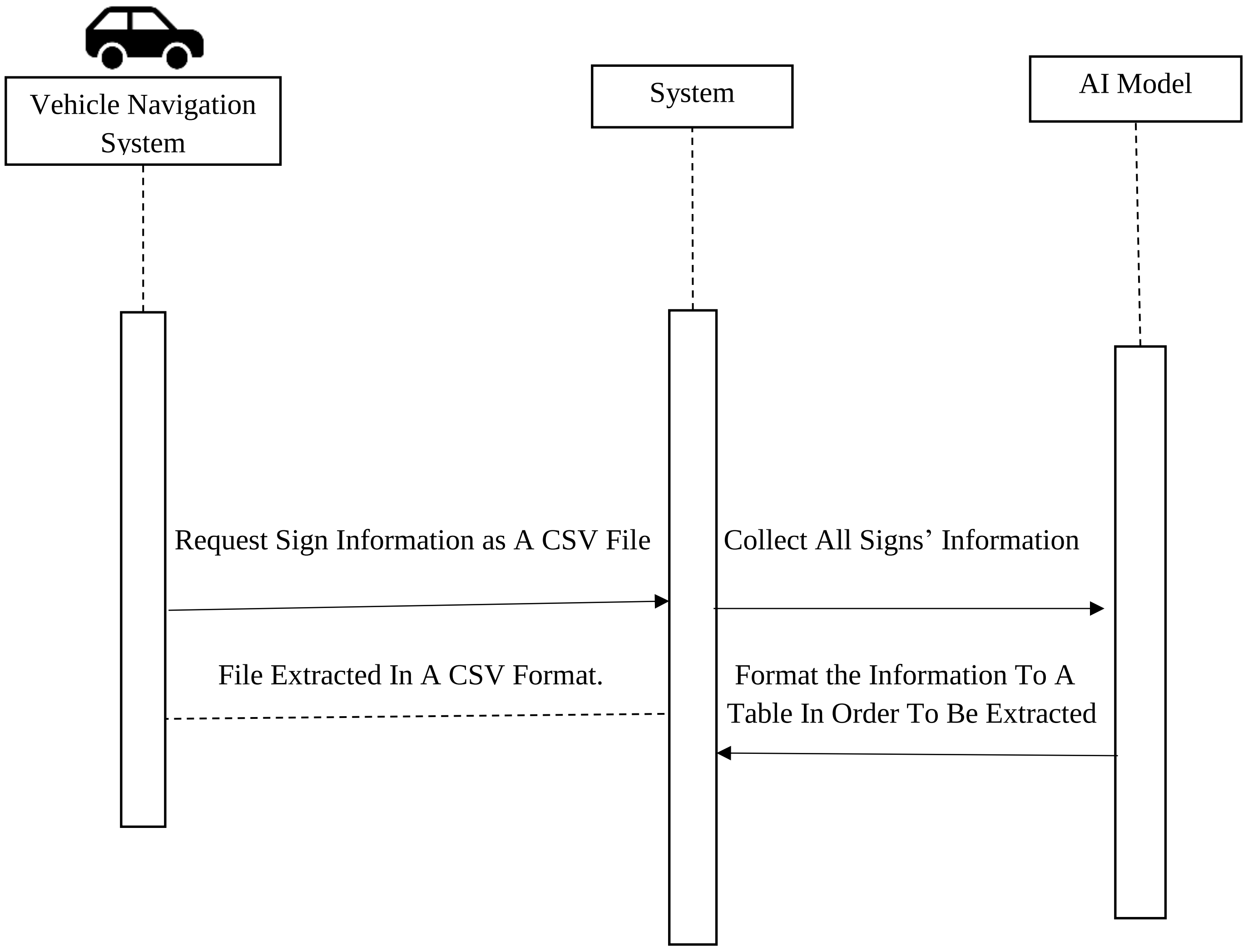

Vehicle Navigation System
System
AI Model
Request Sign Information as A CSV File
Collect All Signs' Information
File Extracted In A CSV Format.
Format the Information To A Table In Order To Be Extracted

Figure 5.5 - Detection and Plotting Sequence

CHAPTER 6

IMPLEMENTATION

6.1 – Purpose of the project

In the previous chapter we explained what our system architecture is, using different diagrams that describe the role of each module, and actor. In the current chapter comes the implementation of what was previously modeled in an abstract thinking. Basically the titles that are going to be tackled are, the purpose, the hardware & the software elements of our project as well as the advantages and disadvantages of our model, API, and User Interface.

6.1.1 – What to settle for?

As discussed in previous chapters our project uses Python as programming language which is an amazing language for multi-purpose usage such as web development, desktop application, and machine learning and AI. Python is also a cross-platform language that runs on Windows, Linux, and mac-OS operating systems.

6.1.2 – Directions API

During the implementation phase we faced a technical problem (we weren’t able to obtain an API key because it required a credit card only) that caused us not only time but also forced us to divert from using google APIs. Due to this problem we had to use an open sourced API for developers only called openrouteservice.org.

OpenRouteService.org provides many features for developers as an example, it provides a directions API service that aims to demonstrate a close experience to what google provides but all that is at the cost of number of requests made by a single user.

6.1.3 – Street-View API

As for our other need for an API to provide us with a dynamic imaging service of routes at a street level we were unable to find any other service than Google’s that can satisfy this need. With that being said, we tried using the openstreetmap.org street imaging service but it required a lot of effort and experience, as well as proved to be unusable when it comes to providing a 360 degree stable images at the street level view of routes on its map.

6.2 – How did we achieve our accuracy?

In this paragraph we will go into more details on how we actually constructed our model as well as the huge dataset we used provided by INI Benchmark which is an organization that benchmarks data sets that are compiled by the Real-Time Computer Vision group at their institute. The dataset is called "Traffic Sign Detection Benchmark (GTSDB)".

6.2.1 – Convolutional Neural Network

Convolutional Neural Network (CNN) is a deep neural network designed for processing arrays of data like images and picks up the patterns like lines, circles gradients, usually used in computer vision like image classification and recognition. CNN is composed of multiple layers of Artificial Neurons (ANs) and then input image to the network each layer takes a patch and apply the weights and activation function.

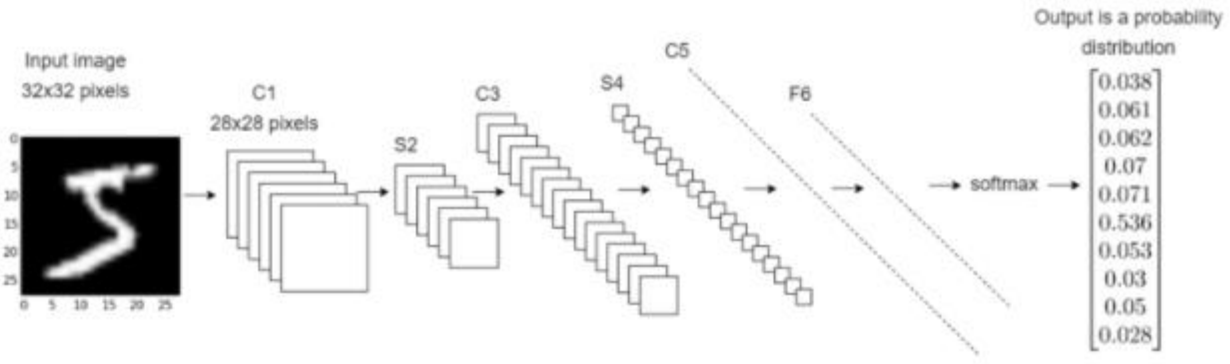
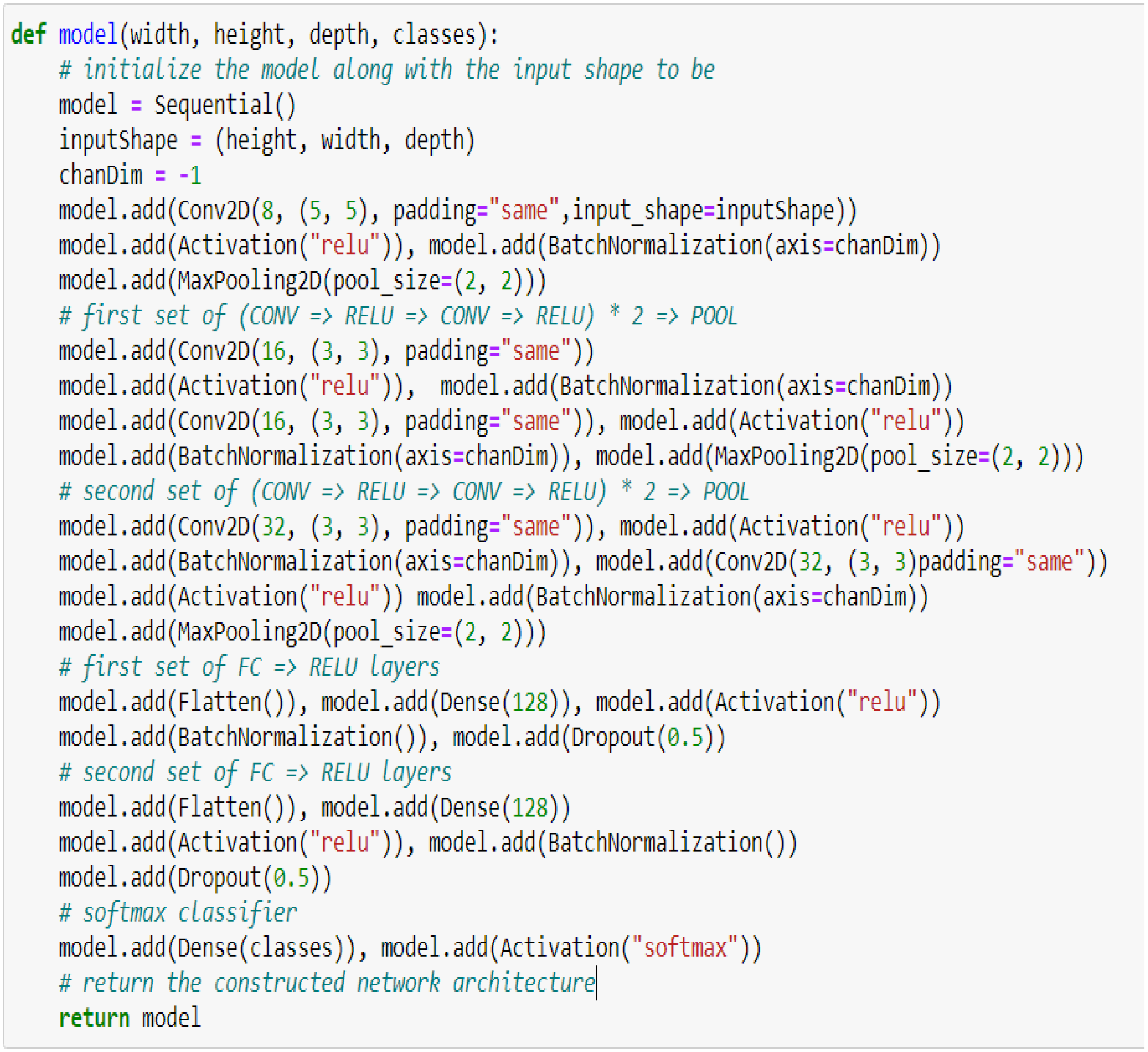


Figure 6.1 – Convolutional neural networks

The figure shows basic LeNet-5 famous CNN where C1 is the first convolutional layer consisting of six kernels of size 5x5 which will walk through the entire image and returns images of size 28x28, second layer is average pooling which will take each four pixels and output the average as one pixel then C2 consists of 16 kernel of size 5x5 which will walk over the outputted images and returns images of size 10x10, and S4 will return the average of each four pixels into one pixel, C5 is fully connected layer which will output 120 nodes connected to the output layer which is of length 10 each node corresponds for digit from 0-9, and the output layer will take the 10 values and run them through Soft-Max activation and return the probability of each node.

6.2.2 – TENSERFLOW.KERAS

Keras is an efficient high level neural network API written in Python which makes it easily understandable and more approachable for fast experimentation. It provides essential abstractions and building blocks for developing machine learning solutions with high iteration velocity. There are two core data structures in Keras, they are models and layers. Keras has a basic model, this model is called a Sequential Model which is used for plain stack layers where each layer has one input and one output, linear models. We will use Sequential because it is easy and efficient for our work.



```
def model(width, height, depth, classes):
    # initialize the model along with the input shape to be
    model = Sequential()
    inputShape = (height, width, depth)
    chanDim = -1
    model.add(Conv2D(8, (5, 5), padding="same",input_shape=inputShape))
    model.add(Activation("relu")), model.add(BatchNormalization(axis=chanDim))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    # first set of (CONV => RELU => CONV => RELU) * 2 => POOL
    model.add(Conv2D(16, (3, 3), padding="same"))
    model.add(Activation("relu")),  model.add(BatchNormalization(axis=chanDim))
    model.add(Conv2D(16, (3, 3), padding="same")), model.add(Activation("relu"))
    model.add(BatchNormalization(axis=chanDim)), model.add(MaxPooling2D(pool_size=(2, 2)))
    # second set of (CONV => RELU => CONV => RELU) * 2 => POOL
    model.add(Conv2D(32, (3, 3), padding="same")), model.add(Activation("relu"))
    model.add(BatchNormalization(axis=chanDim)), model.add(Conv2D(32, (3, 3)padding="same"))
    model.add(Activation("relu")) model.add(BatchNormalization(axis=chanDim))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    # first set of FC => RELU layers
    model.add(Flatten()), model.add(Dense(128)), model.add(Activation("relu"))
    model.add(BatchNormalization()), model.add(Dropout(0.5))
    # second set of FC => RELU layers
    model.add(Flatten()), model.add(Dense(128))
    model.add(Activation("relu")), model.add(BatchNormalization())
    model.add(Dropout(0.5))
    # softmax classifier
    model.add(Dense(classes)), model.add(Activation("softmax"))
    # return the constructed network architecture
    return model
```

Figure 6.2 – Model Building

First, we initialize the Sequential model. Then there is an input layer where it takes images of shape (32,32,3). This layer is convolutional layer of 8 filters and 5x5 kernel with RELU activation and pooling layer of 2x2 size followed by two sets of convolutional layers, first set of 16 filters and 3x3 kernel each followed by RELU activation and then pooling layer, and the second set of

32 filters and 3x3 kernel, then a Dropout layer of 0.5 rate to avoid overfitting. After that comes two sets of fully connected layers of 128 neurons and RELU activation followed by dropout of rate 0.5 and then the output layer which will be 42 neurons (the number of the dataset classes) and with Soft-Max activation so the sum of predicated values from all the neurons in the output layer adds up to one. Lastly, we build the model and compile it using the predefined “model.compile” function from Keras and after that we trained our model on the huge dataset (training progress can be seen in figure 6.3 as a line graph showing the training and validation accuracy as well as loss).

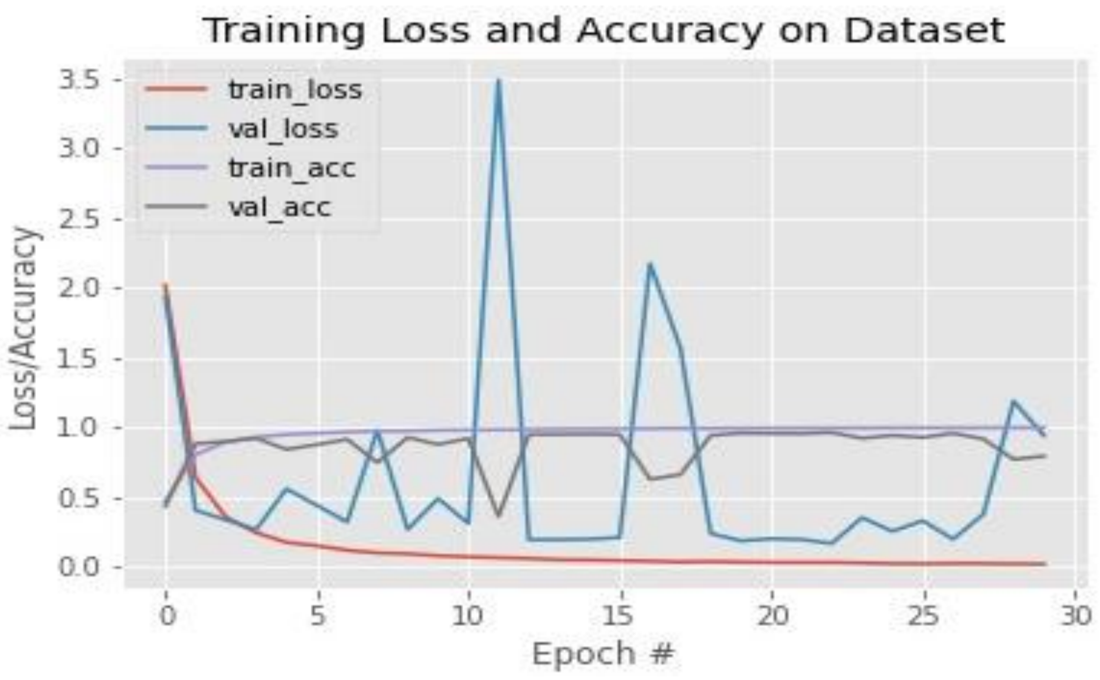


Figure 6.3 – developing the accuracy of the model

After loading the dataset, we resize the images into 32x32 which is the input shape of the model then we apply Contrast Limited Adaptive Histogram Equalization by using “skimage.exposure.equalize_adapthist” which is an algorithm for contrast enhancement to help improve the visuals and details on the image as shown in figure 6.4 using Matplotlib as we constructed two graphs of example sign that got detected using our model. With the help of all the factors above we were able to achieve 99% test accuracy in developing our model.

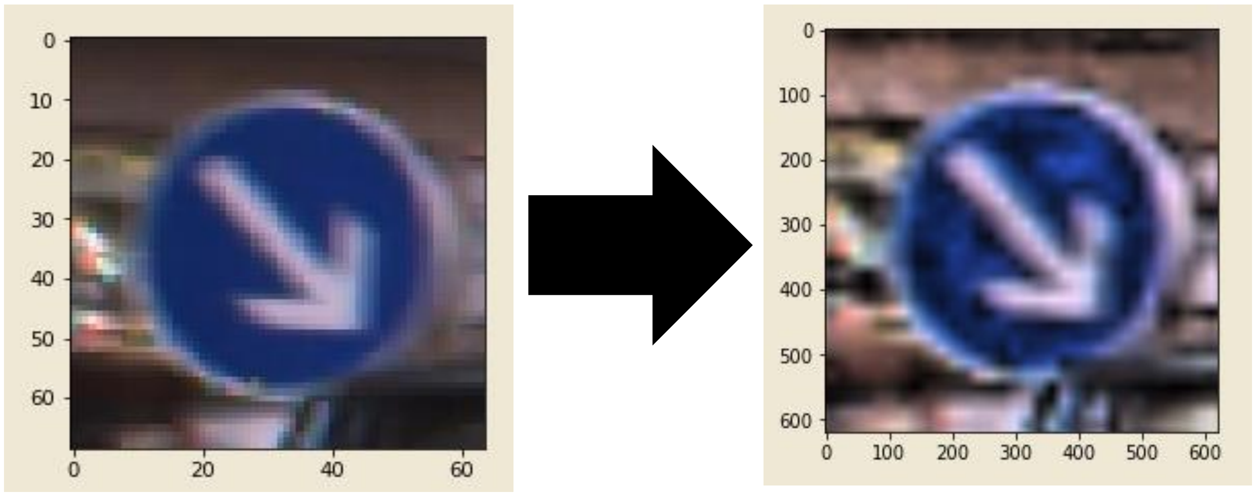
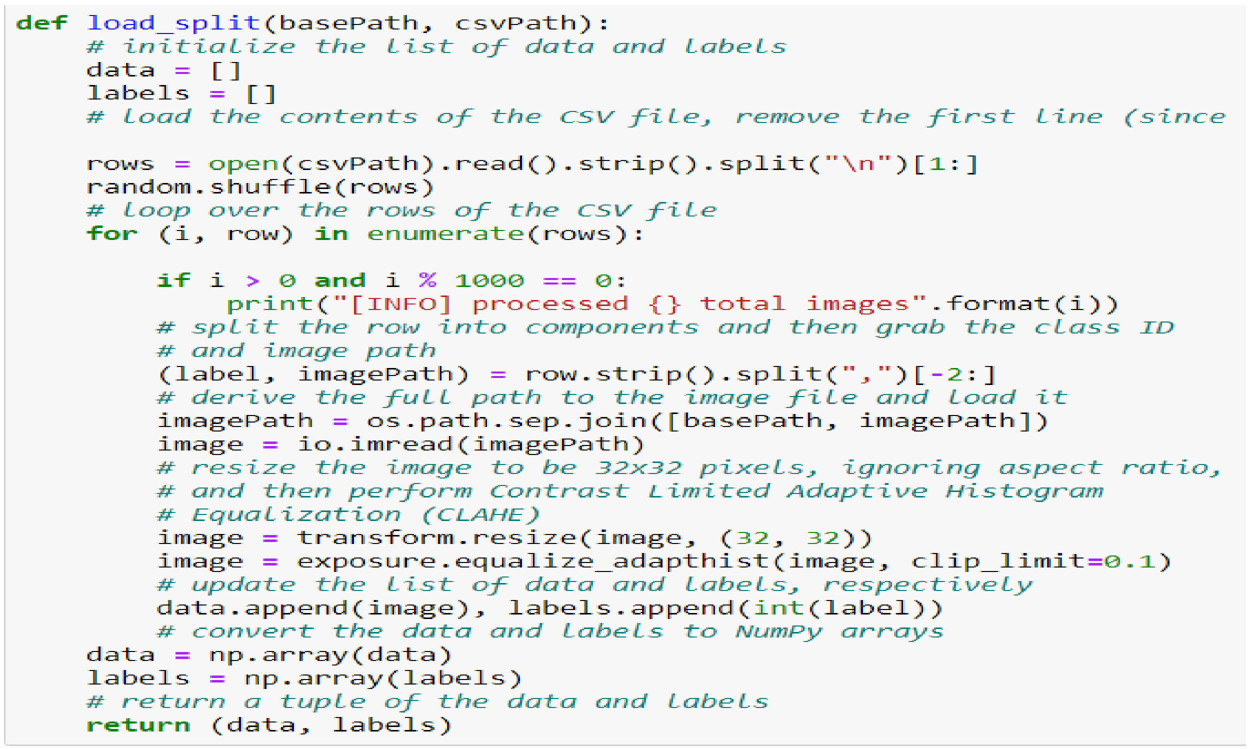


Figure 6.4 – Processing the images

6.2.3 – GTSRB Data Set

The GTSRB[1] (German Traffic Sign Recognition Benchmark) data set was provided by the INI Benchmark institute however it was beautifully prepared to be processed by our model. Figure 6.5 shows the steps we took in order to prepare and use the dataset.



```
def load_split(basePath, csvPath):
    # initialize the list of data and labels
    data = []
    labels = []
    # load the contents of the CSV file, remove the first line (since

    rows = open(csvPath).read().strip().split("\n")[1:]
    random.shuffle(rows)
    # loop over the rows of the CSV file
    for (i, row) in enumerate(rows):

        if i > 0 and i % 1000 == 0:
            print("[INFO] processed {} total images".format(i))
        # split the row into components and then grab the class ID
        # and image path
        (label, imagePath) = row.strip().split(",")[-2:]
        # derive the full path to the image file and load it
        imagePath = os.path.sep.join([basePath, imagePath])
        image = io.imread(imagePath)
        # resize the image to be 32x32 pixels, ignoring aspect ratio,
        # and then perform Contrast Limited Adaptive Histogram
        # Equalization (CLAHE)
        image = transform.resize(image, (32, 32))
        image = exposure.equalize_adapthist(image, clip_limit=0.1)
        # update the list of data and labels, respectively
        data.append(image), labels.append(int(label))
        # convert the data and labels to NumPy arrays
    data = np.array(data)
    labels = np.array(labels)
    # return a tuple of the data and labels
    return (data, labels)
```

Figure 6.5 – Data Preparation Model

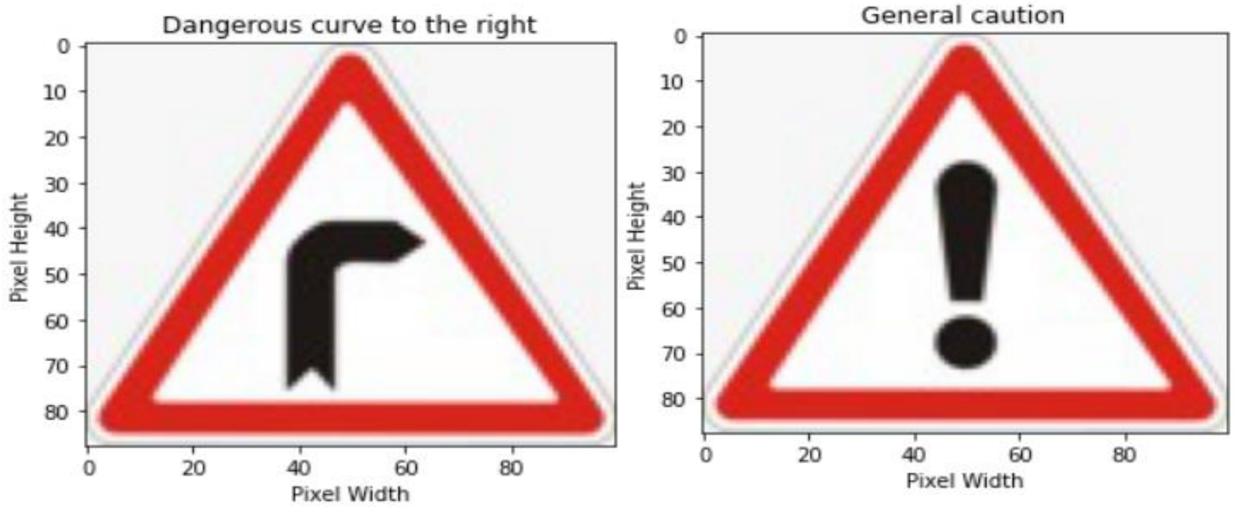


Figure 6.6 – Visualizing Model Predictions

6.2.3 – User Interface Mapping:

In the following paragraph we will go through the application to talk about the user interface that our application has. Our application consists of a single screen that contains a dynamic map as well as a menu of buttons and fields on the left side of the map as shown in figure 6.7 below.

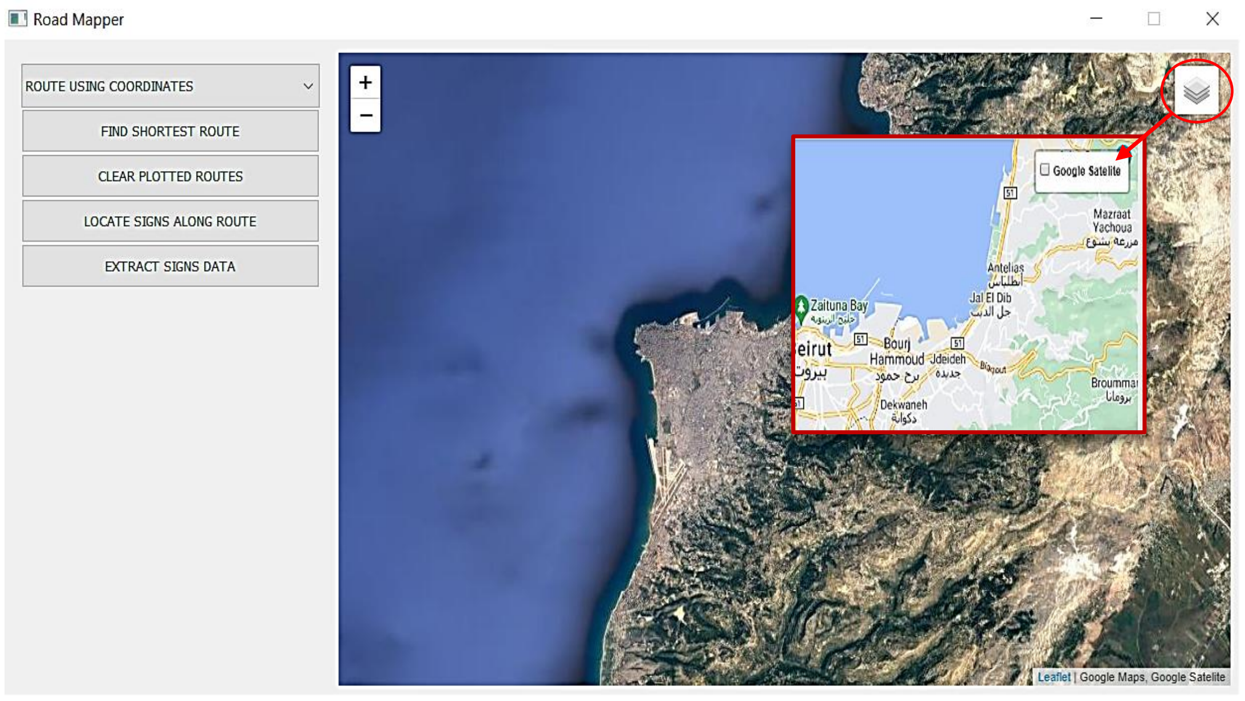


Figure 6.7 – A General Look of the Application

In this paragraph we will go into brief details on what each of our PyQt5 [10] based button does however we will fully dive into them in chapter nine as we test their functionalities. Starting off from the top we can see a dropdown menu button this button lets the user decide which way to enter the location into the application. The application at the moment supports two ways to entering the location either from a place name or from coordinates. Depending on the user’s choice they will be provided with the options to enter a starting and ending locations. The second button is “FIND SHORTEST PATH” this button will take the locations provided by the user and plot a line onto the map indicating the shortest route between the two provided locations. The third button is “CLEAR PLOTTED ROUTES” this button will simply create a new map with nothing on it. The fourth button is “LOCATE SIGNS ALONG ROUTE” this button will take the locations on the map as well as the plotted route then it should check with the google street view API to plot the signs in each image as well as the accuracy of the detection. Now the final button “EXTRACT SIGNS DATA” this button is meant to be used for autonomous vehicles to provide them with the signs directly. So it basically generates a JSON file with the signs detected as well as the location of each sign respectively so the vehicle can use it simulating an API.

6.3 – APIs and Maps Used

In this paragraph we will go briefly into details about the APIs used in the application since there were not that many changes on this side. So the map we used is Folium [9] it is an open sourced map that provides a great extension for Python brought from the leaflet.js map along with this map we put map tiles from google earth engine which are licensed free for research and education purposes. As for our main API for directions we used openrouteservice.org which is an organization that provides a directions API alongside many other features for developers to test and work with. (It should be noted that a huge chunk of time got wasted as we were trying to use openstreetmap.org API in order to do the directions computations in app but that turned out to be inefficient and a waste of time as well as processing power)

CHAPTER 7

SECURITY

7.1 – Security in General

Internet security is a catch-all term for a very broad issue covering security for transactions made over the Internet. Generally, Internet security encompasses browser security, the security of data entered through a Web form, and overall authentication and protection of data sent via Internet Protocol.

7.2 – Security Concepts

Security is generally categorized in four main areas: integrity, confidentiality, availability and authenticity.

7.2.1 – Integrity:

Integrity means that only authorized Admin are able to modify the data when needed, and tampered data is discarded when caught.

7.2.2 – Confidentiality:

Confidentiality means protecting the information from disclosure to unauthorized parties. It ensures that only the authorized Admin can read the information and access the data resources.

7.2.3 – Availability:

Availability summarizes in concept that data should be available and accessible to Admin when needed. Security is needed to prevent any failure to the system.

7.2.4 – Authenticity:

Authenticity is the act of verifying a claim of identity, which means proving the identity of the person you are talking to or proving your identity to a person you are talking to (Admin or customer).

7.3 – Application Security

Now that we took a glance on what the concepts of security are, it is time for us to talk about how we actually implemented them in our application.

7.3.1 – Connections:

Starting off with the security of our connections. Since our application is going to be using google maps' APIs we will go into details on what protection google offers to its API developers.

7.3.1.1 – API Connection Security:

To start off we will explain how google handles API calls.[6] Google APIs only accept requests from registered applications, which are uniquely identifiable applications that present a credential alongside the request. Requests from anonymous applications are not accepted. There are many types of valid credentials that include API keys, OAuth 2.0 client credentials, or service account keys. In our application we chose to use API keys as they are fairly simple and can provide sufficient security [8].

7.3.1.1 – SSL Connection Security:

To identify SSL (Secure Sockets Layer) certificates, they are what enable websites to move from HTTP (Hypertext Transfer Protocol) to HTTPS (Hypertext Transfer Protocol Secure), which is more secure. An SSL certificate is a data file hosted in a website's original server. SSL certificates make SSL/TLS encryption possible, and they contain the website's public key and the website's identity, along with related information. The main company that we will be working with is Google, since they are able to satisfy our needs for both the directions API as well as the street view API. Lastly one of the many reasons we chose to work with Google is that they provide securely encrypted connections to their APIs.

CHAPTER 8

FEASIBILITY STUDY

8.1 – Hardware Requirements

Since Road Mapper is a lightweight Application that does not require any significant heavy processing no matter how many functionalities are added to it, the hardware requirements are minimal and inexpensive. The application has been tested on Windows as well as Mac-OS and can run perfectly with a minimum of 100 MB of RAM and 200 MHz CPU. So, any normal Windows operated machine can run it.

8.2 – Software Requirements

Almost all the used software are free, for the development we used Anaconda and Jupyter notebook for Python which is free, also we used visual studio code. For the map we used Folium which is open-source library for mapping and map views

8.3 – Total Number of Working Hours

The total number of working hours on the project is the sum of the hours spent on all corresponding phases: planning, analysis, design, implementation, and testing.

The planning phase took 6 hours per week by 2 people for 4 weeks, which makes the total time spent on planning is 48 hours. Analysis phase took 4 hours per week by 2 people for 5 weeks making the total time spent for analysis is 40 hours. Design phase took 8 hours per week by 2 people for 6 weeks making total time spent on design is 96 hours. Implementation phase took 8 hours per week by 2 people for 8 weeks making the total time spent on implementation is 128 hours. The total time spent on testing phase is 8 hours per week by 2 people for 2 weeks which making the total time for testing is 32 hours. The total hours spent on the project is 344 hours.

8.4 - Cost Per Hour:

The cost per hour of the development efforts presented by every individual can be estimated to be around the average hourly earnings of a newly hired junior software developer, estimated to be $15 per hour.

8.5 – Overall Cost of the Project:

The overall cost of the project is the price of the culmination of all the resources spent in order to achieve the final product. The total cost of hardware and software requirements is zero since all used requirements are open-source. The total cost of the working hours that the developer team have contributed to develop this application goes as follows in table 8.5.1:

Table 8.5.1 - Phase-based Cost planning

Phase	**Price**
Planning Phase:	720$
Analysis Phase:	600$
Design Phase:	1440$
Implementation Phase:	1920$
Testing Phase:	480$
Software:	0$
Hardware:	0$
Total cost:	**5160$**

CHAPTER 9

TESTING AND VERIFICATION

9.1 – Development and Component Testing

One of the most crucial steps in creating a software is testing the implemented solution. The main purpose for testing is validating that the proposed solution actually solves the problems that it was designed to fix in the first place. Testing process has two simple achievements to obtain, first demonstrating to the developers and customers that the application meets its requirements, secondly discovering all the situations in which the behavior of the application is incorrect, undesirable or does not meet the specifications. In this chapter we will introduce the testing and verification of designed application.

9.1.1 – Development Testing:

Development testing is essentially testing all activities that are carried out by the development team in the program development phase. Development testing is made up of three essential parts. The first part is unit testing, where individual units of the program are tested. The second part is the component testing, when several tested units are integrated to create composite component to help us figure out the compatibility issues that might happen when all units of the program come gather in a single component. Lastly comes the third part which is where the application is tested as a whole.

9.1.2 – Testing Errors:

While testing, lots of errors occur, however the most common type of errors that sneak into the application unnoticeable are interface errors. In our application these errors fall into two main categories. First category is the interface misuse which means pressing a button to do some activity but then doing another activity and generate an error while using its interface. The second category is the timing error; that error occurs in real time system that uses a shared memory or a message-passing interface. It is the action which can make place within sequence of other actions.

9.2 – User Interface Mapping

In the following set of figures we will go through the application screen by screen based on the scenarios that user might go through. Figure 9.1 below shows what the startup screen of the application would look like.

9.2.1 – Startup Page:

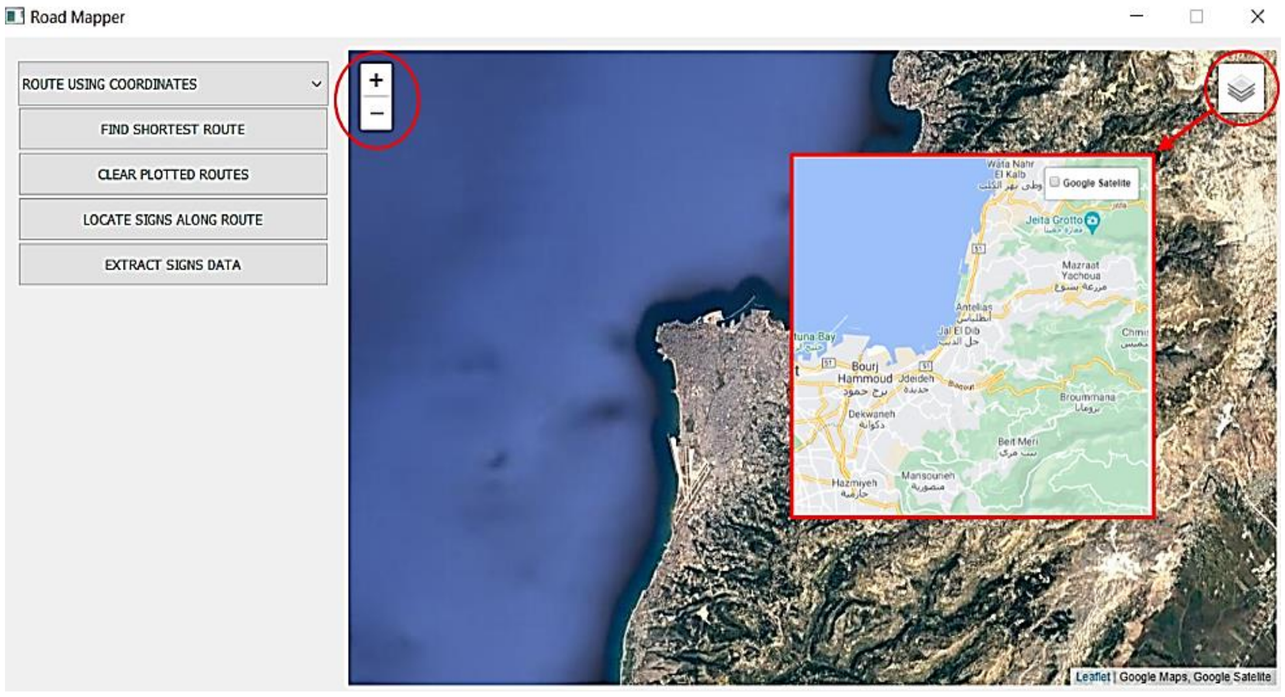


Figure 9.1 – Startup Screen and Map Features

As shown in figure 9.1 the user will have many options starting off with the features of the map. It will have two map tiles the "satellite _maps" as well as the "google _maps" from google earth engine meaning that it will have the ability to be as clear as google maps itself. The user will be able to switch between tiles by clicking the button in the upper right corner of the map. Also in the upper left corner there is a button to help ease zooming in and out of the map which can also be done using mouse wheel.

9.2.2 – Buttons Functionalities:

On the left side of the application we will see the main group of buttons that control the behavior of the application. So let us start by the dropdown menu button at the top. As shown in figure 9.2 once this button is pushed a list of options will appear. In the list there are two options, the first option "ROUTE USING COORDINATES" displays four empty textboxes that tell the user to enter the latitude and longitude of each of the starting point as well as the destination point. The second option "ROUTE USING PLACE NAMES" will display only two textboxes telling the user to insert the starting point's name as well as the destination point's name. In the figure below we will see how the user interface is affected dynamically based on the user's decision.

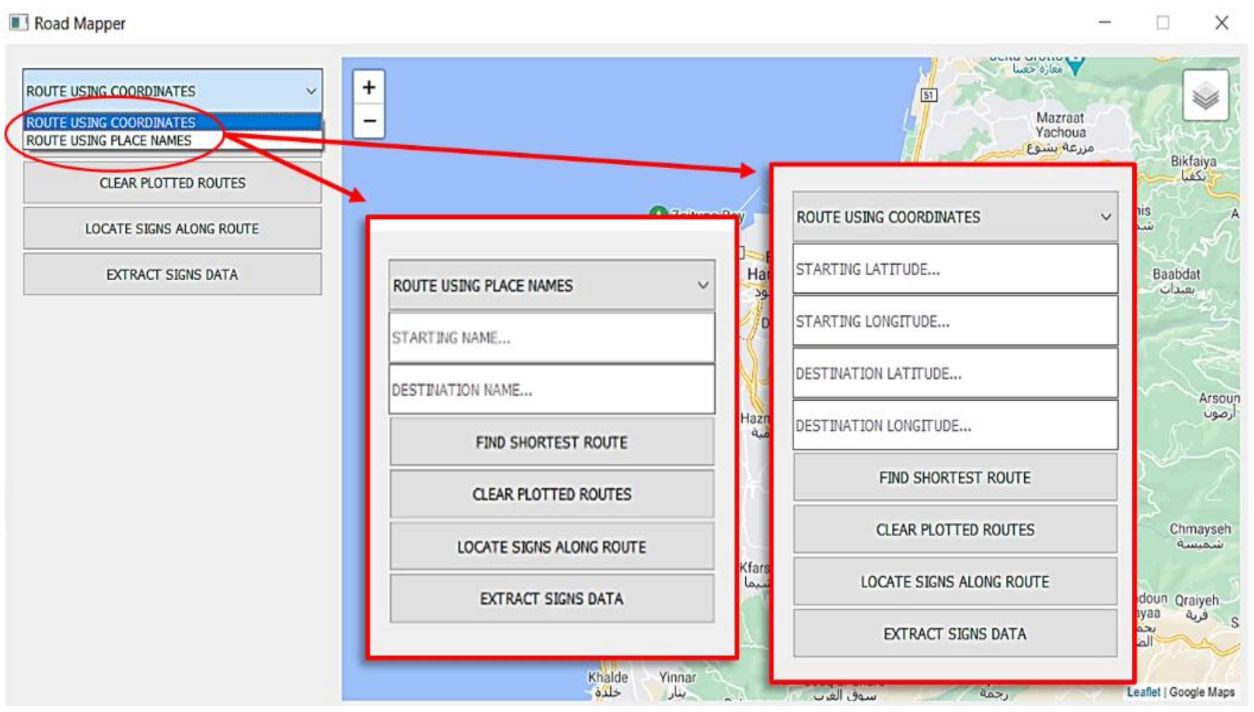
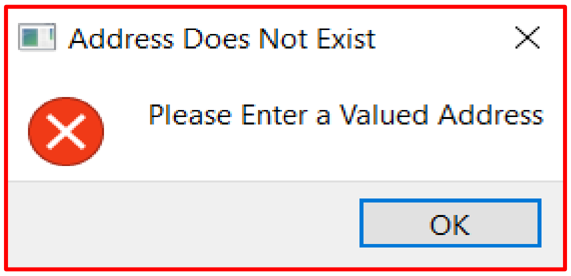


Figure 9.2 – Coordinates Options and map tiles options

9.2.2.1 – ROUTE USING COORDINATES:

If the user chooses this option as shown above in figure 9.2 four text boxes will appear. These textboxes are regex validated to only accept coordinates, meaning you can only enter a number of type double with only two numbers before the decimal point and up to 15 numbers after it for an example “123.123” and “2145.1” are not valued as input however “12.12314124” and “-12.1234124” are valued as input. Figure 9.3 shows the error message discussed in this paragraph.


Address Does Not Exist

Please Enter a Valued Address

OK

Figure 9.3 – Unvalued Address Error Message

9.2.2.2 – ROUTE USING PLACE NAMES:

If the user chooses this option as shown above in figure 9.2 two textboxes will appear telling the use to enter the starting point’s name as well as the destination point’s name. This name is then being validated using the API of openrouteservice.org. If any of the points has an unvalued name the user will be prompted with an error popup message saying that they need to provide a valued place name for the application to be able to show you a route. The error massage discussed in this paragraph is shown in figure 9.3

9.2.2.3 – FIND SHORTEST ROUTE:

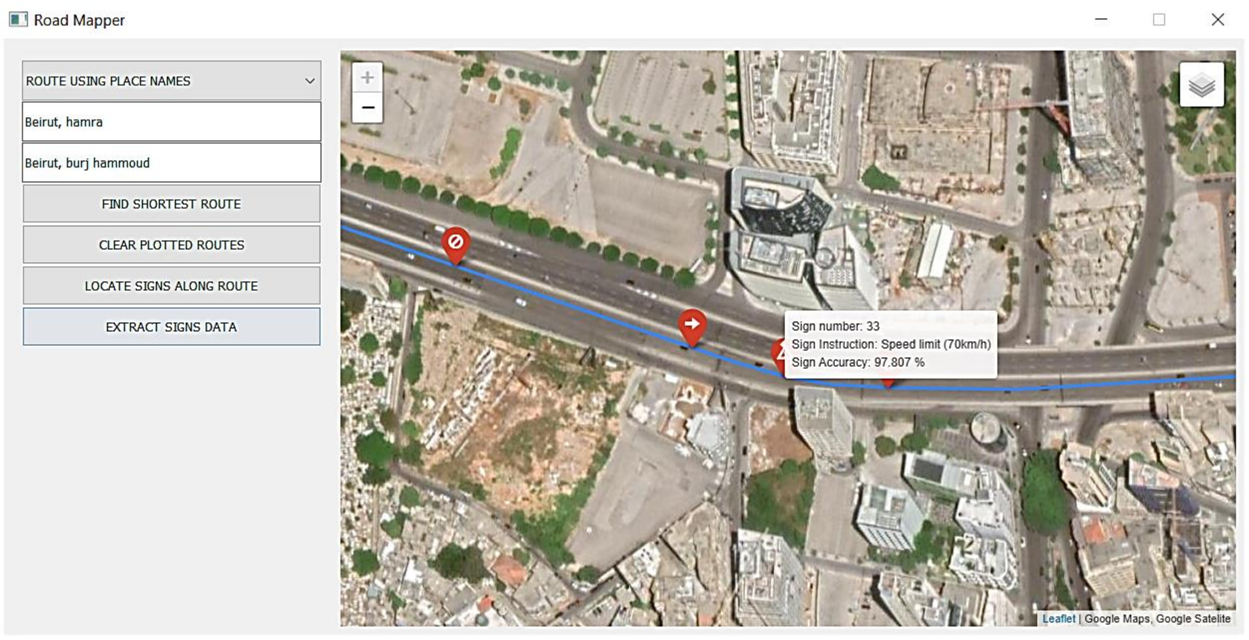
As for this button if the user clicks on it without entering anything in any of the required fields the map will simply refresh, however if the user enters anything in one of the textboxes then we will have two scenarios the first one is where the user enters a valued address in either ROUTE USING PLACE NAMES fields or ROUTE USING COORDINATES fields. In this case four markers as well as a line plot will be displayed on the map. The four markers will indicate the beginning and ending of the closest route to the location entered as well as the starting and ending location points themselves. Along with the markers a route between the beginning and ending markers will be drawn to indicate the shortest route between these locations. However in the other case if any of the addresses or points are unvalued or empty the user will be prompted with a "Please enter a valued address" error message.

9.2.2.4 – CLEAR PLOTTED ROUTES:

When the clear button is pushed in any scenario it simply refreshes the map and creates a new one.

9.2.2.5 – LOCATE SIGNS ALONG ROUTES:

When this button is clicked also a couple of scenarios will happen hence, in this paragraph we will talk about what each scenario is. This button will take the locations on the map as well as the plotted route then it should check with the google street view API to know if the plotted route supports google street view or not. If it does, then we will go to this route and take a picture every ten meters then run it through our model to detect the signs in it. Then it plots these signs on the map as well as the instructions and the accuracy of the detected sign (it should be noted that we were not able to obtain an API key from google to demonstrate the features of this button therefore in the demo we showed what it is meant to look like only. We did NOT implement it in our Demo Application).

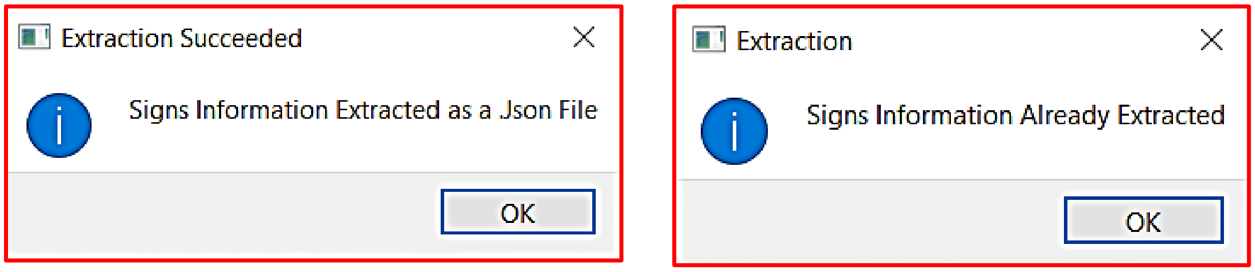


Sign number: 33
Sign Instruction: Speed limit (70km/h)
Sign Accuracy: 97.807 %

Figure 9.4 – Locating Signs Screenshot

9.2.2.6 – EXTRACT SIGNS DATA:

When this button is pushed the first time with a located route the user will be prompted with a message indicating that a new JSON file has been created with the sign names as well as the location coordinates of each sign and if the user clicks it again it would not create the same file another time but it would simply say that a file has already been created and the user needs to plot another route in order for it to register a new route. On the other hand if the user presses this button with no route plotted on the map then the map will simply refresh and nothing will happen. Figure 9.5 shows the different errors described in this paragraph.


Extraction Succeeded

Signs Information Extracted as a Json File

OK

Extraction

Signs Information Already Extracted

OK

Figure 9.5 – Error Messages When Extracting Sign Information

CHAPTER 10

CONCLUSIONS AND FUTURE WORK

Page

10.1 - Summarization

After going through the long and tiresome process of development, it was obvious that the achieved result and gained knowledge could only be reached after hard work. As stated earlier, Road Mapper is an application that provides a better and safer solution for Autonomous Vehicles to perform better. However, despite the magnitude of these results, there is always room for improvement, and planned future functionalities are always present. This chapter concludes the findings and the testing in paragraphs, and plan for future work on the application.

10.2 - Conclusions

Road Mapper provides a safer and more accurate solution with near real-time detection to avoid as much accidents as possible, with a simple user interface where the user can choose either the coordinates or the name of his location.

10.3 - Acquired Knowledge

There are many lessons learnt during the development of this application. One of the most important lessons is teamwork. Learning how to work in team means leaving your comfort zone to work and coordinate with others. Another lesson learned is how to market your application. As technical minded individuals, we learned how to explain the application to people that do not possess our knowledge. This was very hard to do. In addition we learnt how to create GUI application in python using pyQt5 and how to plot and manipulate maps using folium, also we learnt how use Keras to build neural networks for image classification, and many libraries for image processing and machine learning like open-cv, Image, Ski-mage, Sci-kit learn.

10.4 - Future Work:

As future work for Road Mapper, we may add the following features:

- Configure the system to recognize more objects like building patterns, and road layouts.
- Optimize the model so it will be more reliable.
- We will allow the user to add the location by dropping a marker on the map.

References

[1] Training Dataset:

https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

[2] Google Maps:

https://developers.google.com/maps/documentation

[3] Openrouteservice.org:

https://openrouteservice.org/dev/#/api-docs/directions

[4] Convolutional Neural Network :

https://ieeexplore.ieee.org/abstract/document/8718718

[5] Google Street View API :

https://developers.google.com/maps/documentation/streetview/overview

[6] Google Street View API security:

https://developers.google.com/maps/documentation/streetview/get-api-key

[7] Google Directions API :

https://developers.google.com/maps/documentation/directions/overview

[8] Google Directions API security:

https://developers.google.com/maps/documentation/directions/get-api-key

[9] Folium Map Documentation:

https://python-visualization.github.io/folium/

[10] PyQt5 Documentation:

https://doc.qt.io/qtforpython/contents.html