A Web-Based Personal Driving Assistant Using Real-Time Data and a Dynamic Programming Model

by

Mohammad Ali Alamdar Yazdi

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
August 4, 2018

Keywords: eco-driving, dynamic programming, fuel-optimal control, real-time data, Web application

Approved by

Fadel Megahed, Chair, Assistant Professor of Industrial and Systems Engineering
Aleksandr Vinel, Co-Chair, Assistant Professor of Industrial and Systems Engineering
Lora Cavuoto, Assistant Professor of Industrial and Systems Engineering
Richard Sesek, Associate Professor of Industrial and Systems Engineering
Cheryl Seals, Associate Professor of Computer Science and Software Engineering

Abstract

Eco-driving and eco-routing have recently attracted paramount attention from both academia and transportation industry. Multiple studies have approached this problem from a variety of angles. The goal of the first two parts of this dissertation is to leverage the abundance of freely available real-time data and the advanced computer technology to solve this problem. To this aim, several types of real-time information are required. Unfortunately, there is no comprehensive data resource that contains in one place all these different and independent information. To overcome this hurdle, three data sources of GoogleMaps, OpenStreetMap, and OpenWeatherMap are deployed in this dissertation, from which multiple types of information such as road, weather, and elevations information are data-mined and extracted. The extracted information is then passed to a dynamic programming model which is implemented in Python. The dynamic programming model is constructed to optimize the speed, gear, and route based on forthcoming route information data-mined from different web APIs and online databases. The result of this part of this dissertation is an eco-driving and eco-routing real-time web application. Nowadays, due to the advances in the technology of GPS systems, wearable and portable devices such as smartwatches and smartphones, storing the locations data of a moving entity has become easy and common. GPS locations databases have a large size due to the high sampling frequency, contain a lot of conditionally redundant data and therefore are not well suited for further optimization and statistical analysis. The third goal of this dissertation is to develop an R-based web application that takes in a GPS locations raw database along with two databases of Incidents and Accidents, if they are available, and automates the following: construction of a new database that resolves the aforementioned issues of the original database; generation of interactive visualizations that illustrate different statistics of the trip-based database; enriching the constructed trip-based database with the weather conditions (data-mined from National Oceanic and Atmospheric Administration) that a specific trip encountered; and enriching the

newly created trip-based database with the information of Accidents and Incidents that occurred during a specific trip.

Acknowledgments

This dissertation is dedicated to my family for their constant encouragement and unconditional support.

I would like to acknowledge and thank my advisor, Dr. Fadel Megahed, for his enthusiastic and continued support throughout my graduate studies, and for training me to become a researcher. I would also like to express my great appreciation and gratitude to Dr. Alexandar Vinel for his support and help during the last two years of my Ph.D. I would like to thank the rest of my committee members, Dr. Richard Sesek, Dr. Lora Cavouto, and Dr. Cheryl Seals for reviewing my work and providing valuable and constructive feedback. My thanks also go to Auburn University for its excellent academic service and financial support.

This material is based upon work supported in part by National Science Foundation under Award No. 1635927. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Table of Contents

Abstract ii

Acknowledgments iv

1 Introduction 1

1.1 Problem Description 1

1.1.1 Problems One and Two 1

1.1.2 Problem Three 1

1.2 Dissertation Objectives 2

1.3 Motivation 3

1.4 Significance 5

1.5 Dissertation Layout 6

2 DRIIVE: A Real-time Eco-Driving and Eco-Routing Web Application 7

2.1 Introduction 7

2.2 Modeling and Methodology 9

2.2.1 DRIIVE 9

2.2.2 Fuel Consumption Model 12

2.2.3 Dynamic Programming 16

2.2.4 Benchmark Policy 18

2.2.5 Model Parameters 20

2.3 Conclusion 23

3 Advanced DRIIVE 24
3.1 Introduction 24
3.2 Modeling and Methodology 25
3.2.1 Dynamic Speed Limit and Traffic Signs 25
3.2.2 Weather Information 38
3.2.3 GPS, Voice Activation, and Visual Guide of DRIIVE 38
3.2.4 Enhancing DRIIVE 41
3.3 Web Application 42
3.3.1 Home Page 42
3.3.2 About Page 43
3.3.3 Log-in Page 44
3.3.4 Add Vehicles Page 45
3.3.5 Dashboard Page 46
3.3.6 Navigate Page 52
3.3.7 Summary Page 55
3.3.8 DRIIVE Page 56
3.3.9 Graphs Page 58
3.4 Results and Discussions 61
3.4.1 Intrastate 61
3.4.2 Interstate 63
3.4.3 Effect of Road Information and Weather Conditions 64
3.4.4 Testing Road Selection Algorithm 67
3.4.5 Sensitivity to Dynamic Programming Parameters 68
3.4.6 Sensitivity to Action Parameters 71
3.4.7 Comparison with Existing Methods in the Literature 71
3.4.8 Discussion 72

3.5 Conclusion and Future Recommendations 73

4 An R-based Web Application for Mining GPS locations data 75

4.1 Introduction 75

4.2 Modeling and Methodology 78

4.2.1 Locations Databases 78

4.2.2 Online Databases 79

4.2.3 Trip-Based Database 81

4.2.4 Algorithm 83

4.3 Web Application 85

4.3.1 Home Page 85

4.3.2 Upload Page 85

4.3.3 Historical Analysis Page 90

4.3.4 Real-Time Analysis Page 99

4.4 Results and Discussion 101

4.5 Conclusion and Future Works 102

5 Conclusion and Summary of Dissertation Contributions 104

References 107

List of Figures

2.1 Summary of DRIIVE Navigation Model. This figure illustrates the interactions between user inputs, Google Maps API's, Dynamic Programming Model and cruise control Model. The inputs are a pair of origin and destination addresses. The outputs are Dynamic Programming guidelines and cost function for both cruise control and dynamic programming models. 10

3.1 A "node" example is shown by a red-filled circle. A "node" is the most fundamental data-element of the OpenStreetMap database. 28

3.2 A "way" example is shown by a solid line in color orange. A "way" is an element type in the OpenStreetMap database. 30

3.3 A "relation" example is shown by a very long (more than 2,000 "nodes") solid line in color orange and a circle. A "relation" is the most general element type in the OpenStreetMap database. 31

3.4 Importance of angle in road finding algorithm. If angle is disregarded, it is possible that the incorrect road r_2 is returned. Left: Intersection. Right: Highway Exit 34

3.5 The first step of road selection algorithm (RSA): a search area (the outer square area in this figure) is constructed based on endpoints of a given route and a tuning parameter β. If the given location is within that area, the given road is selected as a candidate and passed to the second step of RSA. Otherwise, it is disregarded. 35

3.6 The DRIIVE page has *visual guide* (top left circle filled with color blue, currently showing an arrow to the left (West direction)), a *GPS* (an icon in color blue shown on the bottom right side of the map, and a *voice* feature (the blue left box in the middle left side of the page). 38

3.7 Wireframe for page "Home" 43

3.8 A snapshot of the final version of page "Home." 43

3.9 Wireframe for the page "About" 44

3.10 A snapshot of the final version of the page "About". 44

3.11 Wireframe for the page "Log-in" 45

3.12 A snapshot of the final version of the page ”Log-in”. 46

3.13 A snapshot of the final version of the page ”Add Vehicles”. 46

3.14 Wireframe for the page ”My Vehicles” 47

3.15 A snapshot of the final version of the page ”My Vehicles”. 47

3.16 Wireframe for the page ”My Routes” 48

3.17 A snapshot of the final version of the page ”My Routes”. 49

3.18 Wireframe for the page ”Edit My Info” 49

3.19 A snapshot of the final version of the page ”Edit My Info”. 50

3.20 Wireframe for the page ”Change Password” 50

3.21 A snapshot of the final version of the page ”Change Password”. 51

3.22 Three SQLite tables of ”Dashboard” database and their fields. 51

3.23 Navigate Front and back-ends. A schema of the front and back-ends of page ”Navigate” and how they interact with each other and with the rest of the web application. 54

3.24 A snapshot of the final version of the page ”Navigate”. 55

3.25 A snapshot of the final version of the loader animations in the page ”Navigate.” 55

3.26 A snapshot of the final version of the page ”Summary”. 56

3.27 A snapshot of the final version of the page ”DRIIVE”. 59

3.28 A snapshot of the final version of the page ”Graphs” page. Once the web application is used for an origin-destination pair, a user can select, from the drop-down menu on the top-left corner of the page, up to six routes to simultaneously compare their performances over time in terms of velocity, gear, throttle, brake, acceleration, road grade, and distance through the course of the trip. When the cursor hovers over one point in time, the exact values of selected routes appear next to the cursor, simultaneously. A button on the top-right corner of this page allows users to download all these data for further analysis. For this figure, two routes are selected by a user. 60

3.29 Auburn, AL to Franklin, GA. There are six sub-figures corresponding to velocity, gear, brake, acceleration and road’s grade profiles for a 10-mile portion of the route. 62

3.30 Summary Page for Auburn, AL to Opelika, AL. Three routes and their travel time, distance, MPG, fuel consumption are illustrated for the dynamic programming model. 63

3.31 A Summary of Three Routes for Auburn to Atlanta 64

3.32 Comparison of cruise controller and Dynamic Programming Results for a Route from Auburn to Atlanta 64

3.33 Auburn to Atlanta - Second Try. Two routes are returned, instead of three routes shown in Figure 3.31. This is because the web application takes real-time information. 65

3.34 The Web Application Identifies a Traffic Signals 67

3.35 The Web Application Identifies a Highway Exit 68

3.36 The Web Application Misses One of the Stop Signs 68

4.1 All locations of one truck for a given day are extracted from the GPS locations database and shown on the map using small green circles. Also, trips made by that truck for that day along with their destinations are illustrated by thick solid red lines and red stars, respectively. 81

4.2 A screen-shot of the homepage of the developed R-based web application. This page provides a brief introduction to the project, its goals, and its developers. It also contains a short video on how to use this web application. 86

4.3 Database upload options - Step 1 86

4.4 Users may upload an already created trip-base database. 86

4.5 A snapshot of the slider panel located on the left side of the web application page. On its upper part, the slider panel has links to all pages of this web application, and on its lower part, a summary of the generated trip-base database is shown, which contains information on the number of trips, incidents, accidents, vehicles, drivers, and length of the time period covered in the database. 87

4.6 The raw database can be in a CSV file format or a MySQL format. 87

4.7 For MySQL format, users should fill four fields of User, database, password, and table. 88

4.8 As part of the upload process, users are asked to match the required columns to their uploaded raw database. 88

4.9 A progress bar showing the percentage of the constructed trip-base database. . . 89

4.10 Uploading incident and accident raw databases 89

4.11 Trip data tab of the Historical Analysis Page. Data are illustrated in the format of a table. The table by default is composed of 22 columns. The user has the option to remove any column or sort the table based on any column. We have designed the left side panel such that it can be hidden if the user wants to see the table in full width. 90

4.12 The information in the trip-base table is shown on an OpenStreetMap using the leaflet library for R ([1]). This visualization is interactive allowing a user to zoom in and out, and interact in general with the map. The map has six components of green lines, dark shaded squares, blue dots, yellow circles, brown circles, and red circles. 92

4.13 Hourly intensity of moving trucks. For each hour on the x-axis, the number of trucks moving during that hour is shown on the y-axis. 93

4.14 Daily intensity of moving truck-hours. The x-axis is the x^{yh} day of the year, and its corresponding y coordinates represent the total number of moving truck-hours. 94

4.15 Weekly intensity of moving truck-hours. The x-axis is the x^{yh} week of the year, and its corresponding y coordinates represent the total number of moving truck-hours. 94

4.16 Monthly intensity of moving truck-hours. The x-axis is the x^{th} month of the year, and its corresponding y coordinates represent the total number moving truck-hours. 95

4.17 Profile of hourly intensity of moving trucks for each day of the week. For example, the y-coordinate of graph Monday corresponding to hour x is the sum, over all Mondays of our trip-base database, of total number of moving trucks during hour x. 95

4.18 Histogram of the trip time. 96

4.19 histogram of the stop-elapsed-time. 97

4.20 The tools panel which is located on the left side of the Historical Analysis page consists of six sets of tools 98

4.21 A Snapshot of the Real-Time Analysis Page. The blue circles of the map on the right side of the figure are the GPS locations that are updated every minute in real-time. 100

4.22 Summary box gets updated as new trips are identified in real time. On the right side is a snapshot of the Summary box taken a few minutes after the left one is taken. 101

List of Tables

2.1 Gear parameters (taken from [2]). LEST and UEST are respectively the lower and upper engine speed threshold 21

2.2 Vehicle parameters (taken from [2]). 22

2.3 State sets. The third column is the vehicle's speed range specific to a gear of the second column. 22

3.1 Speed Limits. The first column is the set of the road types (way tag with k="highway") in the OpenStreetMap database. The second column is the set of the speed limits we used for a road type of column one when speed limit (value (v) of the way tag with k="highway") is not specified in the OpenStreetMap data for a specific road. 37

3.2 Weather conditions. The first and second columns are respectively the code and description of different weather conditions in the OpenWeatherMap condition database. The third column is the multiplicative factor by which a speed limit is adjusted for a specific weather code. 39

3.3 Visual Direction. Keywords are given priority for look-up. 40

3.4 Values of Action Parameters and Objective Function Coefficients 61

3.5 Fuel Consumption for a Route from Auburn, AL to Franklin, GA 61

3.6 Auburn to Atlanta with Different Weather Conditions 65

3.7 Effect of Road Information on Fuel Saving 66

3.8 Sensitivity Analysis to the Dynamic Programming Parameters. CC: cruise controller, DP: Dynamic Programming. Columns Fuel, Time, V Change, and Brake are the coefficients in the objective function of dynamic programming. For a cruise controller model, there is no objective function. Hence no coefficients are specified. TT: Travel Time, MPG and Gallon are the outputs of the model corresponding to each set of parameters. 70

3.9 Sensitivity Analysis of Route Selection to Action Parameters. CC: cruise controller, DP: Dynamic Programming. Column Mass is the mass of the vehicle. TT: Travel Time, MPG and Gallon are the outputs of the model for a vehicle with specified weight. 71

Chapter 1

Introduction

1.1 Problem Description

1.1.1 Problems One and Two

It is well documented in the literature that road transport contributes significantly (20% in the year 2014, [3]) to the total CO_2 emissions worldwide. In [4], six main factors for saving fuel consumptions are identified, from among which, roadway-related and driver-related factors are reported in [5] as the most influential, followed by travel-related and weather-related factors. As published in [5], these factors can save up to 40% of fuel consumption.

Eco-driving and eco-routing are defined as a set of decisions, including driving-behavioral decisions, that mitigates the environmental impact of fuel consumption [6]. Eco-driving and eco-routing require real-time road conditions data and real-time weather-related data. Thanks to the advanced technology, nowadays, these data are mainly available to the public online. The problem remains to develop a platform in order to collect and clean these real-time data sets, combine them and use them in solving eco-driving and eco-routing problems in real-time, and to further integrate it to an eco-friendly cruise controller. The first two parts of this dissertation are focused on this problem.

1.1.2 Problem Three

Over the past decade, big data has been the center of attention in both academia and industry, mainly due to the advances in the technology, and due to scientific innovations in the Electronic field that have significantly reduced the storage cost of very large-scale data. GPS locations data is an example where big data is applicable. GPS locations databases, in their raw format, are

too large and contain a lot of conditionally redundant data due to the high frequency of location sampling. Furthermore, these databases are not trip-based. Because of the aforementioned issues inherent to them, these databases are very difficult to be used for data mining and statistical/optimization analysis. The third part of this dissertation is to develop an R-based web application that helps to solve the aforementioned issues.

1.2 Dissertation Objectives

The first goal of this dissertation is to develop a web application (DRIIVE) that leverages the abundance of freely available real-time data and the advanced computer technology to solve the eco-driving and eco-routing problem in real-time using dynamic programming.

The second goal of this dissertation is to advance DRIIVE such that: 1) It takes the speed limits of routes into consideration; 2) it takes real-time weather information into account in order to increase the safety of driving; 3) it provides new features that enable the users to specify their own vehicle, to indicate how much fuel consumption saving is important to them in a trade-off with the overall trip time, and to determine how much driving action they expect; and 4) a tracking GPS system featured by voice and visual guide is added to the web application to make it easier and much safer for the drivers to use.

The third goal of this dissertation is to develop an R-based web application that takes in a GPS locations raw database and automates the following: (1) construction of a new database that resolves the inherent issues of the original database; (2) generation of interactive visualizations that illustrate different statistics of the trip-based database; (3) enriching the constructed trip-based database with the weather conditions that a specific trip encountered; and (4) enriching the newly created trip-based database with the information of Accidents and Incidents that took place during a trip.

1.3 Motivation

Based on multiple studies (see [5], [7]), improving infrastructure and adopting eco-friendly driving behavior, and eco-friendly routing schedule are some means of reducing fuel consumption. As the authors in [8] also argue, while the first method can save significantly more fuel consumption relative to the latter two approaches, the insignificant (to none) extra capital cost of eco-driving and eco-routing speaks to the importance and worthiness of these two approaches. Furthermore, although eco-driving and eco-routing fuel consumption savings per trip seem small, it mounts to very noticeable amounts over the whole population of the country and over a wider time horizon.

Eco-driving and eco-routing are defined in the literature as a set of decisions, including strategic, tactical and operational, that mitigate the environmentally harmful impact of fuel consumption [6]. Strategic decisions are the ones on vehicle's class, model, configuration, and maintenance. These strategic decisions have a significant influence on the fuel consumption. For example, a good vehicle model can be 9 times more fuel-efficient than a bad vehicle model [6]. However, since strategic decisions of eco-driving are financially costly, the focus of this research, like the studies of [9, 10, 11], is on the tactical decisions (eco-routing) and operational decisions (driving behavior).

The Velocity Trajectory Optimization problem, in which fuel consumption, travel time, and road information are considered, has been extensively studied in the literature [12]. Different approaches have been introduced to attack the velocity trajectory optimization problem. Examples of such approaches are Pontrygain's Minimum Principle [13], Indirect Multiple Shooting Method [14], Sequential Quadratic Programming (SQP) [15], and dynamic programming [2, 16, 17]. In our research, we use the dynamic programming approach because of its fast Johspeed in solving the problem and its high accuracy.

The GPS locations databases are commonly generated and maintained for different applications. These databases are very large and contain a lot of information. This information, once data mined, can be used and analyzed for different purposes. For example, transportation companies can use such databases to increase transportation efficiency, reduce transportation

fuel cost, advance routing schedule, and increase firm's commercial profit and customer's satisfaction. In addition to the commercial value of this information to the firm, such information can be beneficial to traffic control, urban development, and policymaking. This information can even be used for individual people. As mentioned before, unfortunately, there are some inherent issues with such databases in their raw format that makes them unsuited for data analysis. It is of paramount importance to overcome these issues and construct new databases well suited for data mining. This is the aim of the third part of this dissertation.

The problems we are addressing in this dissertation are appealing to multiple industries and academia.

The first industry is the Delivery industry. Delivery companies, and in general, successful firms, have to be financially profitable and maintain a good level of customers satisfaction at the same time. The customers satisfaction requires delivery companies to be on-time in delivering products to their customers. Any delay in delivery will add to customers dissatisfaction, which will consequently lead to less business for the Delivery company and hence less financial profit. To be on-time, delivery companies need to spend more money on their delivery system. Any saving in fuel consumption of each delivery will mount to a significant amount of saving over the course of a year. Therefore any optimization of fuel consumption is financially beneficial to the Delivery companies. The second industry is the transportation industry in which fuel saving of each trip accumulates to a large amount over thousands of trips the companies' trucks have over a year, which translates to a financial benefit for the firm.

The web application we have developed for the first problem allows both transportation and delivery companies to find the optimal speed profile that suits their purposes best. For example, if a delivery company needs to deliver a package as fast as possible for customer satisfaction, they can use our web application such that they find the speed profile that gives them the shortest travel time. On the other hand, if the delivery company can tolerate a longer travel time, they can also use our web application and find the optimal speed profile such that they meet their travel time tolerance and at the same time have saved some fuel which translates to financial profit for the firm.

Transportation companies can also use the second web application we have developed to better understand their driver's trip characteristics. Analyzing the trip profile of each driver help transportation companies in their trip allocations algorithms and, in general, in their policymaking. It is of paramount importance for transportation companies to be able to know what trips have high crash risk, in which areas and in what periods of time; to be able to know if accidents and incidents occur in trips with longer travel times or with shorter travel times; to be able to know if there is any abnormality in the trip profile of a specific driver relative to the others; to be able to identify locations where most of the trips have stops. The web application we have developed in our second research problem provides transportation companies with a new database that is well-suited for such statistical and optimization analyses.

1.4 Significance

The existing solutions to the eco-driving problem assume the required data are readily available, which is not a practical assumption. Furthermore, some of these solutions provide some general guidelines based on some general assumptions that ignore the real-time information and therefore may not result in much fuel consumption saving in some cases. Our web application, on the other hand, uses real-time information about the road and weather conditions. Therefore, our guidelines are tailored to the specific road and weather conditions. Our developed web application has been designed such that it can be used in real-time. Furthermore, our web app is accessible to the public for free.

As for the third goal of this research, to the best of our knowledge, this is the first free and accessible-by-public R-based web application specific to the GPS locations databases. The newly created database (that is the output of our developed web application) is very valuable for further use in data mining and analysis. The output database can be readily used for vehicle-routing applications. In addition, the visualization outputs of our developed web application are interactive and very informative. Our web application and R-based web application are capable of taking both real-time and historical GPS locations databases.

1.5 Dissertation Layout

In this dissertation, three objectives are identified and developed. These objectives are specific to data mining applications and fall into the broader realm of big data. This dissertation is organized as follows: the first objective is to develop a real-time eco-driving web application (DRIIVE). DRIIVE, at this stage, will data-mine information from the freely available GoogleMaps databases to internally solve a dynamic programming model for eco-driving and eco-routing applications. In Chapter 2, DRIIVE is developed and described in details. Chapter 3 provides a comprehensive description of the development of advanced DRIIVE which is the second objective of the dissertation. In Chapter 4, an R-based web application for dealing with GPS locations databases along with a web application is developed and described.

Finally, this dissertation is concluded in Chapter 5, in which the contributions of this dissertation are summarized and some future recommendations are made.

Chapter 2

DRIIVE: A Real-time Eco-Driving and Eco-Routing Web Application

2.1 Introduction

Fuel consumption can be appealing to environmentalists, economists, and car manufacturers. But more specifically, the resultant fuel consumption of driving is economically important to individual drivers. It is reported in [16] that an average truck driver drives more than 150,000 km per year. Therefore, any fuel consumption reduction, as small as it can be, would lead to significant cost savings for drivers over the course of multiple years, and accumulated over a multi-million population.

Driving fuel consumption is dependent on a variety of factors, some examples of which are vehicle's type, road conditions, weather conditions, and driving behavior. All these factors can be categorized into two main super-factors that are independent of each other, yet their interaction highly influences the driving fuel consumption.

The first super-factor is the vehicle's type and characteristics that affect the fuel consumption. This is probably the main deciding factor, which is in the hands of manufacturers and not individual drivers. There are different classes of vehicles with known fuel consumption range. Vehicle brands also decide on their fuel consumption. In the same row is vehicle's age, and size. We do not focus on this super-factor, but instead, treat it as a set of external parameters.

The second super-factor is the speed profile of driving. Speed profile of driving refers to the profile of speeds, accelerations, and gears a vehicle drives on, and how much brakes it takes during a trip. In fact, the effect on driving fuel consumption of driving behavior, weather conditions and road conditions all get reflected in the speed profile of driving; this super-factor is highly dependent on the driver's driving habits and reactions to real-time external and internal

factors, such as weather and road conditions, and traffic. This super-factor is the focus of this research.

The speed profile super-factor effect on fuel consumption has been studied in the literature extensively. Many researchers have approached this problem from different angles and by making different assumptions. Finding a solution to this problem requires solving a Velocity Trajectory Problem (VTP).

A VTP finds the ”best” speed profile such that some cost function is minimized, under some conditions and constraints. Examples of known approaches in the literature to the VTP are Pontrygain's Minimum Principle [13], Indirect Multiple Shooting Method [14], Sequential Quadratic Programming (SQP) [15], and dynamic programming [2, 16, 17, 12]. We have chosen the dynamic programming approach in our research due to its efficiency.

To find a solution to the problem of speed profile super-factor, one needs to have the road topology of the route. There have been several researchers who have either collected the topology of a limited number of roads from sensors, received the data from other sources or made some assumptions [17, 16]. In [13] the VTP under constant road grade assumption has been found. Rather than making assumptions, we have built a platform in our research problem that collects the information of road topology from online resources and use the data to solve the VTP.

The focus of this chapter is to develop a real-time eco-driving and eco-routing Web Application (DRIIVE) that internally solves the velocity trajectory optimization problem using a dynamic programming model. Our web application uses Google Maps API ([18]) real-time information to find a fuel-efficient speed profile under some time constraints for a given trip.

The rest of this chapter is as follows. In Section 2.2, all models, methodologies, and model parameters used in DRIIVE are described. We postpone the software development description of our application and the results and discussions sections to chapter three. Section 2.3 concludes this chapter.

2.2 Modeling and Methodology

The focus of this section is the models and the methodologies that have been used in this chapter. In the first subsection, we describe the general picture of the DRIIVE application without going deep into the implementation details of its components. The second subsection is the fuel consumption model based on which the fuel consumption of a trip is calculated. This model is taken from the work of [2] and the parts that are most relevant to our research are re-described. The dynamic programming model that is used to solve the trajectory optimization problem is described in the third subsection. This model is described in the context of our research. Finally, the cruise controller model is described. It has been used as a benchmark against which the results of our dynamic programming model are compared. The cruise controller model is taken from [2].

2.2.1 DRIIVE

In this section, we describe details of our model that uses the available information from Google Maps database and feed them into a dynamic programming model in order to optimize an objective function for given origin and destination addresses. The diagram of this model is shown in Figure 2.1. The user needs to first enter addresses of the origin and destination on the web. The following three options are provided to the user:

1. If the exact address of the origin and destination are known, they can be entered by users.

2. If the exact address of the origin and destination are unknown, a user can enter a partial address and the app will automatically display all related addresses, from among which a user can select the address of interest. To this aim, while typing the address on the web, DRIIVE connects to the Google Maps API and passes the entered partial address to the Place Autocomplete API of the Google Maps. It then lists some addresses that have the best match to the entered address. The user will then have to select the correct address based on the matches.

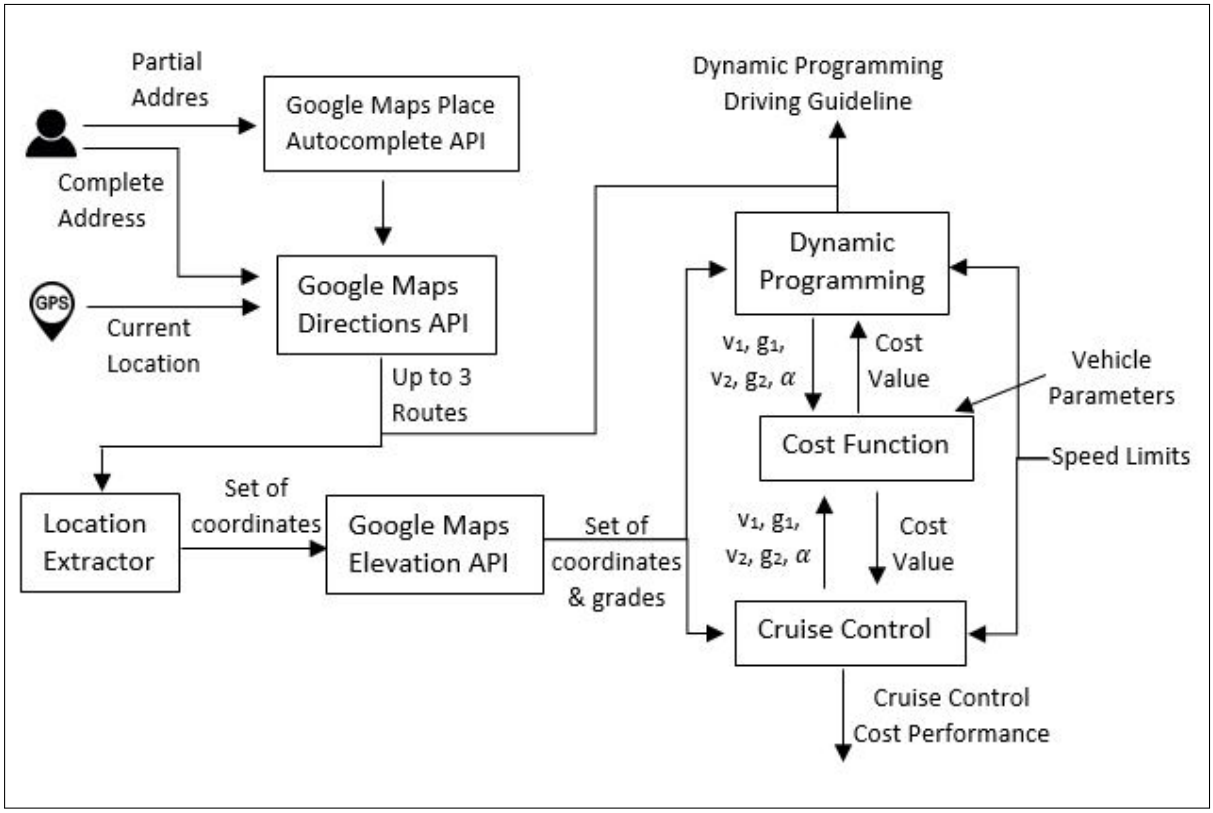

Figure 2.1: Summary of DRIIVE Navigation Model. This figure illustrates the interactions between user inputs, Google Maps API's, Dynamic Programming Model and cruise control Model. The inputs are a pair of origin and destination addresses. The outputs are Dynamic Programming guidelines and cost function for both cruise control and dynamic programming models.

3. Specific to the origin address only, another option for the user is to let DRIIVE app uses the device's GPS to find the current location of the user. This can be done by using the HTML DOM Navigator Geolocation property.

Once the origin and destination are set, the app calls the Google Maps Directions API with these two addresses. The Google Maps API returns a JSON data file containing some information about (up to) three shortest routes. The JSON data file contains step by step driving direction, latitude, and longitude of each point of the route, in addition to some other information.

Each route is defined by a collection of geographical points. For each route and internal to this app, some points are selected such that the distance between two consecutive points is 0.1 mile, and stored as a pair of latitude and longitude. The web passes the extracted latitude and

longitude information of the points of each route to the Google Maps Elevation API; in return, Google Maps Elevation API provides the elevation of these geographical points.

The app then connects to a local server and calls a dynamic programming model using (up to three) arrays of elevations (each corresponding to one route) to solve the optimum routing problem. Fed to the dynamic programming model is also a set of speed limits that is used internally in the dynamic programming model to establish the feasible set of states. The dynamic programming model internally calls a Cost Function multiple times and tries to find the optimal speed and gear such that the overall value of the cost function is minimized. The input to the cost function is also the vehicle parameters that are used for calculation of fuel consumption. The dynamic programming model is performed on each of the (up to three) routes and returns the following outputs.

- Speed profile
- Overall fuel consumption in gallons
- Average fuel consumption per mile
- Total time of the trip

To compare the result of our dynamic programming, we need to have a benchmark driving behavior. There are different driving behavior, the most popular of which is driving on a cruise controller.

Similar to the dynamic programming model, the information of (up to three) arrays of routes along with their elevations, a set of speed limits, and vehicle parameters are passed to a cruise controller model described in 2.2.4. cruise controller model then internally calls a Cost Function multiple times and returns overall fuel consumption in gallons, average fuel consumption per mile, the total time of the trip, and speed profile.

2.2.2 Fuel Consumption Model

For our research, we use the ICEV fuel consumption model of [2]. In this section, we briefly re-describe this model following the terminologies used in [2] (for more details, the reader is referred to [2]).

Consider a truck is moving on a road with slope α. Four main forces of (1) air resistance F_a; (2) gravitational F_g; (3) rolling resistance F_r; and (4) powertrain force generated by the motor F_w are applied to this truck. The first three forces are external, and the last force is internal.

- $F_r(\alpha)$: for any rolling object, there is a rolling resistance force that is applied to the object opposite to the direction of moving, and is calculated as $F_r = c_r.m.g.\cos(\alpha)$ where m is the mass of the truck and c_r is the rolling resistance coefficient. This force slows down the truck in both uphills and downhills. As this force increases, vehicle's engine needs to generate more power to overcome the increase in this force, which requires more fuel. Based on the above formula, an increase in the truck's weight and an increase in the slope of the road result in an increase in the rolling resistance force, which will consequently increase the fuel consumption.

- $F_g(\alpha)$: thanks to the gravity, there is a gravitational force applied to the vehicle and is calculated via $F_g(\alpha) = m.g.\sin(\alpha)$. This force increases the truck's speed in downhills but decreases it in uphills. Therefore, in downhills, the effect of higher weight is in fuel consumption reduction, but in the uphills, the effect is opposite. Also, as this force increases by slope, in downhills, the higher the slope is, the less fuel consumption will be. Opposite effect exists in the uphills case.

- $F_a(v)$: for any moving object, there exists an air resistance force applied to it. This air resistance force is in the direction opposite to the movement of the object and is calculated via $F_a = \frac{1}{2}c_a.\rho_a.v^2$, in which c_a is an air-related coefficient (called Drag Area), ρ_a is air density coefficient, and v is the truck velocity. Based on this formula, as velocity

increases, this force increases proportionally to the square of the velocity. Unlike previous forces, this force does not depend on truck's weight or road's slope and depends only on the truck's speed and vehicle's design. The higher the truck's speed, the higher the fuel consumption of the truck will be. This force is one of the main causes of fuel consumption,

- F_w: the powertrain force which helps the moving truck reaches its target acceleration. This force is a function of engine, transmission, driveline, and wheel.

Based on Newton's second law of motion, the equivalent force of these four forces and the mass of the truck m determine the acceleration of the truck, $\dot{v}$ via:

$$\dot{v} = \frac{F_w - F_a(v) - F_r(\alpha) - F_g(\alpha)}{m} \tag{2.1}$$

Equation 2.1 can be rewritten as the following equation:

$$\dot{v} = c_1(c_3 T(v,P,G) - k_b B - c_2 v^2 - c_6 sin(\alpha + c_7)) \tag{2.2}$$

where

$$c_1 = \frac{r_w}{J_w + m r_w^2 + \eta_t(G) i_t(G)^2 \eta_f i_f^2 J_e} \tag{2.3}$$

$$c_2 = \frac{1}{2} c_a \rho_a r_w \tag{2.4}$$

$$c_3 = \eta_t(G) i_t(G) \eta_f i_f \tag{2.5}$$

$$c_4 = \frac{30 i_t(G) i_f}{\pi r_w} \tag{2.6}$$

$$c_5 = \frac{n_{cyl}}{60000 n_r} \tag{2.7}$$

$$c_6 = 9.81 * mr_w\sqrt{1+c_r^2} \tag{2.8}$$

$$c_7 = \arctan c_r \tag{2.9}$$

and

- J_e is the mass moment of inertia of the engine,
- n_{cyl} is the number of cylinders of the engine,
- n_r represents the number of crankshaft revolutions per stroke,
- N_{idle} is the engine idle speed which is in revolution per minute,
- δ_{idle} is the idle fueling
- i_f and η_f are respectively the final drive ratio and final drive efficiency,
- J_w is the wheel inertia,
- r_w is the wheel radius,
- $k_b$is the brakes maximum torque,
- $i_t(G)$ and $\eta_t(G)$ are respectively the ratio and efficiency coefficients of gear G.

Total fuel consumption, m_f, can be obtained by solving equations 2.2 to 2.14. N denotes the engine's speed in RPM, $\hat{T}(v,P,G)$ is the torque generated for a velocity v, gas signal P and gear G, $\dot{m}_f$ is the rate of engine's fuel consumption, and $\hat{\delta}_{max}$ is the fuel flow upper limit.

$$N = \begin{cases} c_4 v & G \neq 0 \\ N_{idle} & G = 0 \end{cases} \tag{2.10}$$

$$\hat{\delta}_{max}(N) = a_\delta N^2 + b_\delta N + c_\delta \tag{2.11}$$

where a_δ, b_δ, c_δ are the fuel flow limit coefficients,

$$\hat{T}(v,P,G) = \begin{cases} a_e N + b_e P \hat{\delta}_{max}(N) + c_e & G \neq 0 \quad P > 0 \\ a_d N + b_d & G \neq 0 \quad P \leq 0 \\ 0 & G = 0 \end{cases} \tag{2.12}$$

where a_e, b_e and c_e are the coefficients of engine map, and a_d and b_d are the coefficients of drag torque.

$$\dot{m}_f(N,P,G) = \begin{cases} c_5(NP\hat{\delta}_{max}) & G \neq 0 \\ c_5(N_{idle}\delta_{idle}) & G = 0 \end{cases} \tag{2.13}$$

$$m_f(N,P,G) = \int \dot{m}_f(N,P,G)dt \tag{2.14}$$

Note that c_1 to c_7, N_{idle}, δ_{idle}, a_δ, b_δ, c_δ, a_e, b_e, c_e, a_d, and b_d are vehicle parameters.

Gear Selection

Engine speed N, gear G and vehicle speed v cannot be selected free of each other. More specifically, the Engine speed N is bounded by some lower and upper thresholds $N_{lower}(G)$ and $N_{upper}(G)$ which are a function of gear G. The function that maps G to $N_{lower}(G)$ and $N_{upper}(G)$ is specific to the vehicle's gearbox. This restricts the gears that the vehicle can be driving on for a specific value of vehicle speed. To find the gears in which a vehicle with speed v can operate, we do the following:

1. Calculate engine speed N based on current gear G and vehicle speed v via formula 2.10,
2. if the speed calculated in the previous step is within $N_{lower}(G)$ and $N_{upper}(G)$ thresholds, return current G and exit. If not, go the next step.
3. Increase current gear to $G+1$ if the speed N calculated in step 1 is greater than $N_{upper}(G)$ threshold, and go to step 1.
4. Decrease current gear to $G-1$ if the speed N calculated in step 1 is less than $N_{lower}(G)$ threshold, and go to step 1.

2.2.3 Dynamic Programming

The fuel consumption optimization problem is mathematically a continuous (in location and speed) optimization problem. However, to solve this problem researchers treat this problem as a discrete optimization problem, and reach out to algorithms that are advanced for addressing these kinds of problems. More specifically, one can discretize the route between origin and destination into N locations, and further discretizes the possible vehicle speed into M speeds, and find the optimal set of speeds (of size N) each corresponding to one location such that the overall objective function of fuel consumption is minimized.

Researchers have used algorithms that can solve such problems iteratively.One of these algorithms is Dynamic Programming (DP) which, since invention, has been extensively studied, implemented and tested for a large variety of applications.

In the context of our research:

- each stage of DP refers to a physical location of the vehicle in a route. If the route is composed of N locations, the DP has N stages.
- At each stage i, $1 \leq i \leq N$, the speed and gear on which the vehicle is operating define a state of DP at that stage.
- Decision variables: throttle and brake
- Objective function: a linear combination of travel time, fuel consumption, absolute change in speed, and brake level.
- Constraints: the speed at each stage must be less or equal to the road speed limit v_{max}. In some roads especially highways, there is minimum speed v_{min} which drivers are not allowed to drive slower than that. Therefore, the speed at each stage must fall between given speed limit range $v_{min} \leq v \leq v_{max}$. To provide a safe driving guideline, we should avoid hard acceleration. A maximum a_{max} and a minimum a_{min} acceleration levels are considered as safe driving thresholds. Dynamic programming should not consider an acceleration out of this range as a feasible solution at any stage, i.e., $a_{min} \leq a \leq a_{max}$.

For mathematical conciseness, let's denote by $S_f^k = \{x_i\}_{i=k}^N$ a feasible set of states $x_i = (v_i, g_i)$ from stage k to stage N, i.e., (v_i, g_i) is a feasible vehicle speed and gear at stage i. Note that the vehicle speed must be between an upper and a lower speed limit that is dictated by the road type, traffic information, weather information, etc., and therefore not all v_i's are feasible. Also, not all combinations of (v_i, g_i) are feasible. This is mainly due to the engine characteristics and functionality. For each range of vehicle speed, there is only limited number of possibilities for the gear. These restrictions should be considered to decide whether a state (v_i, g_i) is feasible.

To each S_f^k there is a cost $Z(S_f^k)$ associated, which is basically the total cost from the location of stage k to the end of the road if the vehicle obeys the speeds and gears dictated by S_f^k. Such cost is calculated via:

$$Z(S_f^k) = Z_N(x_N) + \sum_{i=k}^{N-1} z(x_i, x_{i+1}) \tag{2.15}$$

where $z(x_i, x_{i+1})$ is the linear combination of fuel consumption m_f required to go from state x_i to state x_{i+1}, time t_i required to go from state x_i to state x_{i+1}, absolute change in speed $|v_i - v_{i+1}|$, and brake level B_i required to go from state x_i to state x_{i+1}. The formula for z is shown below:

$$z(x_i, x_{i+1}) = w_1 \frac{m_{f,i}}{n_1} + w_2 \frac{t_i}{n_2} + w_3 \frac{|v_i - v_{i+1}|}{n_3} + w_4 \frac{B_i}{n_4} \tag{2.16}$$

in which w_1 to w_4 are the predefined weights, and n_1 to n_4 are nomralization factors, which are take values of 1, 12, 20 and 100 in this research.

The aim is to solve the following optimization problem:

$$arg \min_{S_f^0} Z(S_f^0) \tag{2.17}$$

where the minimization is conducted on all possible S_f^0.

To solve 2.17, DP divides the problem into N smaller optimization problems and solve them recursively. More specifically, the backward DP does the following:

1. Let's define x_i as the state variable of DP in stage i and x_{ij} as the j^{th} realization of state x_i in stage i. We then define $S_f^{\star i,j}$ as the set of x_m, $i \leq m \leq N$ for stages of i to N, that minimizes $Z(S_f^i)$ under the constraint that $x_i = x_{ij}$.

2. Variable Initialization: $i \leftarrow (N-1)$, $S_f^{\star N,1} \leftarrow \{(v_N, g_N)\}$

3. For each k, $1 \leq k \leq n_i$, (with n_i being the number of states at stage i)

   - Solve for

   $$\bar{j} = arg \min_{1 \leq j \leq n_{i+1}} Z(\{(v_{ik}, g_{ik})\} \cup S_f^{\star(i+1),j}) \tag{2.18}$$

   - Set: $S_f^{\star i,k} \leftarrow \{(v_{ik}, g_{ik})\} \cup S_f^{\star(i+1),\bar{j}}$

4. Set: $i \leftarrow (i-1)$

5. if $i > -1$, go to step 3, otherwise return $S_f^{\star 0,1}$ as the optimal set of states, and $Z(S_f^{\star 0,1})$ as the optimal value of the objective function.

2.2.4 Benchmark Policy

There are different driving behaviors, the most popular of which is driving on a cruise controller. For testing purposes, we pick a cruise controller driving behavior.

It is straightforward and common knowledge what the purpose of a cruise controller is: once a target speed limit is set, the cruise controller manages the moving vehicle to reach that target and to maintain the vehicle's speed around the target until a new target is given or the brake is taken. The internal mechanism of how a cruise controller works, however, is more sophisticated and much less known to the public. In the following, we briefly explain a cruise controller model with two proportional-integrals (PI) (similar to the work of [2]) and with an anti-windup feature.

Given current speed of v, the target speed of $v_{target} > v$, and a threshold speed, a cruise controller increases the vehicle speed by accelerating, in short periods of time, until the vehicle speed reaches v_{target}. If during this process, the vehicle speed happens to exceed v_{target} the cruise controller does not react until the vehicle speed passes $v_{threshold} = v_{target} + \delta$ where the

parameter δ is fixed and specific to the type of the cruise controller. If $v > v_{target}$, the cruise controller then applies the brake until the vehicle's speed slows down to below v_{target}

A Cruise Controller System With Two Proportional-Integral Controllers

Consider two variables of brake(B) and fuel allowance (G) both of which are between 0 and 1, with 1 being the maximum possible brake or fuel allowance. These two variables quantify the brake amount or allowed fuel. The purpose of this section is to describe a model that has been used to calculate these two variables.

The model used for each of these two variables is the proportional-integral controller which is a well-established control loop feedback model. For each variable, two parameters are used which require tuning for practical purposes. We denote these two parameters of variable B by B_P, B_I and those of variable G by G_P, G_I. For the following formula, $B(k)$ and $G(k)$ are the values of the two signals B and G at discrete time k, with k starting from 1, and the initial values are $I_G(0) = I_B(0) = 0$.

$$g(k) = G_P \times (v_{target}(k) - v(k)) + G_I \times I_G(k-1) \tag{2.19}$$

$$G(k) = \min(\max(g(k), 0), 1) \tag{2.20}$$

$$I_G(k) = I_G(k-1) + (v_{target}(i) - v(i)) \tag{2.21}$$

$$b(k) = B_P \times (v(k) - v_{target}(k)) + B_I \times I_B(k-1) \tag{2.22}$$

$$B(k) = \min(\max(b(k), 0), 1) \tag{2.23}$$

$$I_B(k) = I_B(k-1) + (v(i) - v_{ref}(i)) \tag{2.24}$$

Integral Windup

It is well known that in a PI controller, if the output signal (in our case B and G) has some upper/lower bounds (saturation points), then the PI controller will react very slowly in the case when the signal is well-saturated, i.e., well-passed its boundary. This is called integral windup phenomena. Integral windup is an undesired side-effect of the integration term of PI

controller [19]. This adverse effect is even more pronounced and harmful in the case of a two PI's controller.

To give an example, at time k, let's assume the vehicle has been going uphill for a long period of time and has not reached its target speed, yet its speed is very close to the target point. In that case, the calculated $I_G(k)$ is much larger than one. Now assume, the slope of the road ahead is less than current slope. At the next step (time $k+1$), the gas signal $G(k+1)$ should be lowered (than $G(k)$), but because $I_G(k)$ is well saturated (is much larger than one), $G(k+1)$ will still remain equal to one, which means the gas level at $k+1$ remains the same as that of time k. Hence, the speed at time $k+1$ will well pass the target speed, which is not desired.

There are several anti-windup methods, from among which we use the following method for a two PI controller, in which the value of $I_G(k)$ resets to zero if $B(k) > 0$, and the value of $I_B(k)$ resets to zero if $G(k) > 0$.

The calculated G and B via formula 2.19 to 2.22 are then substituted in formula 2.12 and 2.13 to calculate the fuel consumption of a cruise controller.

2.2.5 Model Parameters

In order to apply the proposed algorithm, the following sets of parameters must be specified for each trip.

- Route Directions: this information is required to guide the driver as to which direction the car should move on. We collect route directions from Google Maps Direction API.

- Road Topology Data: a road related information that is needed in dynamic programming is the grade (elevation) of the road. The DP result is very sensitive to this information. We collect road elevations from the Google Maps Elevation API and use them to calibrate the grade of each road.

- Vehicle-Related Data: the parameters that characterize the engine, driveline, wheel, gearbox, cruise control system, brake, and vehicle resistance of our vehicle are needed for the cost function. This information is specific to the underlying vehicle and changes from vehicle to vehicle. The detail of such information is as follows:

Gear	Efficiency	Ratio	LEST	UEST
1	0.88	11.27	600	1800
2	0.89	9.14	1300	1800
3	0.90	7.17	1300	1800
4	0.91	5.81	1300	1800
5	0.92	4.62	1300	1800
6	0.93	3.75	1300	1800
7	0.94	3.01	1300	1800
8	0.95	2.44	1300	1800
9	0.96	1.91	1300	1800
10	0.97	1.55	1300	1800
11	0.98	1.23	1300	1700
12	0.99	1.00	1300	-

Table 2.1: Gear parameters (taken from [2]). LEST and UEST are respectively the lower and upper engine speed threshold

  - Engine: J_e, n_{cyl}, n_r, N_{idle}, δ_{idle}, a_e, b_e, c_e, a_δ, b_δ, c_δ, a_d, b_d
  - driveline: i_f, n_f
  - Wheel: J_w, r_w, c_r,
  - vehicle resistance: c_a
  - brake: k_b
  - cruise control: K_P, I_P, K_B, I_B, $v_{threshold} - v_{target}$
  - gearbox: $i_t(g)$, $n_t(g)$, $LB(g)$: the lower threshold of engine speed as a function of gear, $UB(g)$: the upper threshold of engine speed as a function of gear.

  The default vehicle parameters used in this research are those of a typical SCANIA truck which are shown in Tables 2.1 and 2.2.

- Speed Limit Data: For our dynamic programming model, we need the speed limit of the roads. These speed limits vary by the road type, location, the weather information, traffic information and any unpredicted accidents. Unfortunately, Google does not provide this information to the public free of charge, and therefore, not all users can exploit this information. Therefore, we use a default upper and lower speed limits of 45 $\frac{mi}{hr}$ and 70 $\frac{mi}{hr}$, and therefore, our application at this stage is designed for routes that follow Interstate highways.

Parameter	n_{cyl}	n_r	n_{idle}	n_{idle}	i_f	n_f	J_w	c_r	m
Value	6	2	600	10.44	3.27	0.97	32.9	0.007	25000
Unit	-	-	rpm	$\frac{mg}{stroke}$	-	-	kgm^2	-	$\frac{m}{Kg}$
Parameter	J_e	a_d	b_d	a_e	b_e	c_e	a_e	b_e	c_e
Value	3.5	-0.088	-51.51	-0.0033	9.16	-82.53	-.000159	0.42	-57.05
Unit	kgm^2	$\frac{Nm}{rpm}$	Nm	$\frac{Nm}{rpm}$	$\frac{Nmstroke}{mg}$	Nm	$\frac{mg}{rpm^2stroke}$	$\frac{mg}{rpmstroke}$	$\frac{mg}{stroke}$
Parameter	c_a	p_a	r_w	k_b	K_P	I_P	K_B	I_B	$v_{threshold} - v_{target}$
Value	6	1.29	0.52	20000	1.845	0.2	0.3	0.02	1
Unit	m^2	$\frac{kg}{m^3}$	m	Nm	-	-	-	-	$\frac{m}{s}$

Table 2.2: Vehicle parameters (taken from [2]).

States	**Gear**	**Velocity (mi/hr)**
1-26	0	45-70
27-33	11	45-51
34-54	12	50-70

Table 2.3: State sets. The third column is the vehicle's speed range specific to a gear of the second column.

As previously described in Gear Selection section, not all pairs of vehicle velocity and vehicle gear are feasible. This feasible set depends on the vehicle speed, and vehicle parameters. For a typical SCANIA truck and based on the aforementioned vehicle parameters, the feasible set of vehicle speeds and gears are shown in Table 2.3.

- Dynamic Programming Parameters: internal to the dynamic programming model, there are some parameters that have been hardcoded and are described below. The first set of parameters are w_i's, the weights used as the linear coefficient of the objective function as shown in equation 2.16, with the following values: $w_1 = 1$, $w_2 = 1.0$, $w_3 = 1.0$, $w_4 = 1.0$.

  Another set of parameters are related to the maximum and minimum allowed acceleration of the vehicle and are considered to be $-2\frac{m}{s^2}$ and $+2\frac{m}{s^2}$. These acceleration limits are used in the dynamic programming model to check if a change of vehicle speed from one stage to another is allowed. For instance, if a change of vehicle speed is higher than $2\frac{m}{s^2}$, the new vehicle speed is considered infeasible and therefore is not allowed.

2.3 Conclusion

We have developed a real-time web-based application that uses the Google Maps database and a dynamic programming model to solve and find a speed profile to save fuel consumption. We have described in details all the models and methodologies for this web application.

In the next chapter, we will develop our web application such that it incorporates speed limits of each route, road conditions, the traffic signs, and real-time weather information at the time application is running in order to find an optimal speed profile. We will further enhance our web application in the next chapter so that it allows users to specify their own vehicles, and their preference in terms of trade-offs between the amount of travel time, the amount of fuel consumption, the amount of changes in the vehicle's speed and the amount of brake. To increase the safety of the drivers while using our application, we will add some features to the application that require access to real-time GPS location of the device.

Chapter 3

Advanced DRIIVE

3.1 Introduction

To make our application more practical for city driving, we need to identify the traffics signs and determine the speed limits at each part of the route. Unfortunately, Google does not provide this information to the public free of charge, and therefore, not all users can exploit this information. Hence, we need to data mine another database. The database we selected is OpenStreetMap, which is a database that is created, validated, and accessible by the public, free of charge. This database contains a lot of information about road conditions, among which the speed limit, road types and traffic signs locations of each route are of our interest.

As we describe in this chapter, extracting the routes speed limit and location of traffic signs is a very difficult task. This is mainly due to the fact that the Google Maps database and OpenStreetMap are two totally independent databases which do not have the same database structure and their data fields are not exactly the same. Because of such inconsistencies between the two databases, for a given route in the Google Maps database, one needs to first identify this route in the OpenStreetMap database, and therefore data-mining algorithms are required. In this chapter, we develop such algorithms and use them to first extract the speed limit of a route in Google Maps database from OpenStreetMap database and to second identify all traffic signs in the OpenStreetMap database that are located on a route in Google Maps.

Road accidents are among the major causes of death worldwide [20]. We would like to take safety into consideration. Incorporating weather conditions into our dynamic programming model will increase the safety of eco-routing and eco-driving. To this aim, we collect the real-time weather conditions of the routes, based on which the speed limit of each route

is adjusted accordingly. Google Maps database does not have such information. Hence, we collect the weather condition from the OpenWeatherMap API ([21]). OpenWeatherMap API provides the weather condition for a good number of areas in the world.

Vehicle characteristics such as engine specifications and vehicle mass have significant effects on the fuel consumption. To make our application usable for drivers with different vehicles, we have enhanced our application such that users can specify their vehicle and run the application on their own vehicles. Users can store all this information in their profile (as part of the application) for future use and do not need to enter this information every time. There are other such features added to our web application that we explain in this chapter.

Different drivers have different trade-offs between time and fuel consumption. Some prefer to have a trip with a shorter time; some prefer to have a trip with smaller fuel consumption. Also, some prefer to have to make less driving actions such as accelerating or decelerating during the trip. We have enhanced our application such that a user can enter their preferences into the application and run our web application on their own preferences. These are explained in more details in this chapter.

Finally, we have added to our web application a tracking GPS system featured by voice and visual guide to making it easier and much safer for the drivers to use.

The rest of this chapter is organized as follows. In Section 3.2, we describe in details all models and methodologies used in this chapter. In Section 3.3, the software development of our web application is described. Section 3.4 provides the results and some discussion. Finally, Section 3.5 concludes this chapter.

3.2 Modeling and Methodology

All models and methodologies used in this chapter are described in details in the rest of this section.

3.2.1 Dynamic Speed Limit and Traffic Signs

Roads speed limits and the location of traffic signs such as stop signs, yield signs, and traffic signals are not provided by the Google Maps API. We need this information for our dynamic

programming model. To this aim, we have used OpenStreetMap database. OpenStreetMap is an editable free map of the world whose data is available to the public. It contains a lot of information about the physical locations on the map.

We have downloaded from the OpenStreetMap more than 192 gigabytes of data related to North America and have re-designed a new database that is more useful for our purposes. The information contained in OpenStreetMap is not of the same format as the information in the Google Maps. More specifically, the routes of Google Maps are not readily identifiable in the OpenStreetMap database. To identify a route of Google Maps in the OpenStreetMap database, we do the following. For a given location that Google Maps returns, we find a way in OpenStreetMap that is closest to (and not necessarily includes) this location. From the data related to the closest way, the application collects the speed limit. In the case that speed limit is not available, the application calculates the speed limit from other road conditions. We repeat this for all locations of the route. Having the speed limits of each route at our disposal, we then pass these speed limits to our dynamic programming model, which will be used in creating the states of the dynamic programming model.

Through the rest of this section, we use $R_{Goog} = \{loc_i^{Goog} \equiv (lat_i^{Goog}, long_i^{Goog})\}_{i=1}^{N_r}$ to denote a route and all its points extracted from the Google Maps database, with N_r being the total number of constituent points of the route. To find the speed limit profile of the route R_{Goog} and all the traffic signs existing in this route, we reach out to OpenStreetMap database, hereafter referred to by OSM database. Fortunately, OSP database contains such information. The difficulty, though, resides in the fact that Google Maps points $\{loc_i^{Goog}\}_{i=1}^{N_r}$ do not necessarily exist in the OSM database. In the rest of this section, we describe how based on a given route $R_{Goog} = \{loc_i^{Goog}\}_{i=1}^{N_r}$ extracted from Google Maps database, one can extract the speed limits, stop signs, traffic lights, and yield signs information from the OSM database. The algorithm we use is a heuristic one which is chosen after testing a lot of different algorithms.

Let's first fix the goal of this algorithm. For a given Google Maps route $R_{Goog} = \{loc_i^{Goog}\}_{i=1}^{N_r}$, we would like to return a speed limit profile $S^{osm} = \{s_i^{osm} = (m_i^{osm}, v_i^{osm})\}_{i=1}^{M}$ in which m_i^{osm} is the distance from origin of the route R_{Goog}, v_i^{osm} is the speed limit for road portion of $(m_i^{osm}, m_{i+1}^{osm})$, and M is the size of the speed limit profile.

Note that m_i^{osm} and v_i^{osm} are extracted from the OSM database. Further, the size of S^{osm} and R_{Goog} are not necessarily the same, i.e., M and N could be different.

If there is a stop sign ahead, drivers need to stop completely. On the other hand, there is a high chance that the drivers have to stop completely at the locations of yield signs and traffic signals. In order to be conservative about the safety, we assume that we will be having a complete stop in all three types of locations (stop locations from now on). We would like S^{OSM} to reflect the stop locations. In particular, if there is a stop sign in the underlying route R^{Goog} at location m_j^{osm}, we would add $(m_j^{osm}, 0)$ to the set S^{OSM}. Note this information, i.e., m_j^{osm} has to be extracted from OSM database for a route R_{Goog} from Google Maps database.

Constructing the speed limit profile is further complicated by the very large size (of the order of hundreds of gigabytes) of OpenStreetMap database. Therefore, extracting any information from the OpenStreetMap database based on the Google Maps result requires smart data-mining algorithms. It is therefore of paramount importance that the data mining algorithms be smart and that the OSM database be modified such that information extraction can be done with sufficient accuracy, in an efficient way, and with a fast pace.

OpenStreetMap and Database Construction

As mentioned above, it is very crucial to first construct a new database from OSM database in order for the speed profile extraction algorithm to be able to run fast and efficiently. If such database is not constructed in a smart way, any further data-mining algorithm will take forever to perform and return a result. In the following, we describe the OpenStreetMap database and the new database that we have constructed from OSM database. It is noteworthy that this database does not exist in public and we have created it to be tailored to our research problem, i.e., speed limit profile extraction problem.

OpenStreetMap Database

There are three types of elements in the OSM database.

- Node: the most fundamental element of OSM database is a node. A node represents a physical location with a specific latitude and longitude to which an ID is allocated. A

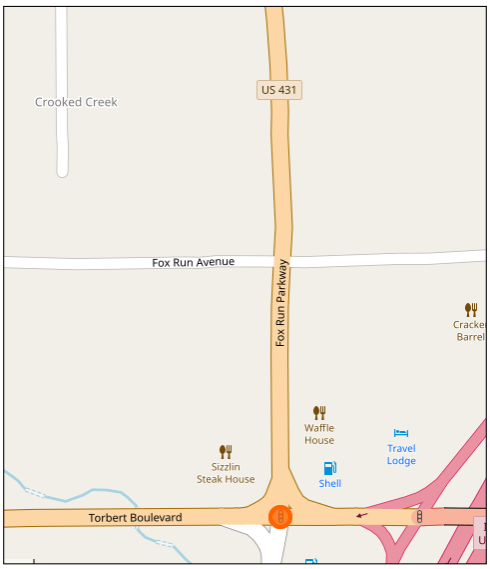

Figure 3.1: A ”node” example is shown by a red-filled circle. A ”node” is the most fundamental data-element of the OpenStreetMap database.

node can potentially have multiple tags. Each tag is a pair of (key, value). For example, a tag can be (highway, traffic signals) which identifies this node as a traffic signal. In addition, a node can have other attributes such as ”visible” (meaning it has been deleted or not), ”version” (showing how many times it has been edited), ”changeset”, ”user”, ”uid” (last three provide some information about the user who added/edited this node), ”timestamp” (the time of the last edit). An example of a node is given in Figure 3.1, with its information in OSM database shown below. This node represents traffic signals with latitude of 32.6436050 and longitude of -85.3549670, and id of 56892768.

```
<node id="56892768" visible="true" version="4" changeset=
"41291034" timestamp="2016-08-06T17:36:48Z" user="Caboosey"
uid="171516"
lat="32.6436050" lon="-85.3549670">
<tag k="highway" v="traffic_signals"/>
<tag k="traffic_signals" v="signal"/>
</node>
```

- Way: a way is an ordered list of 2 to 2,000 nodes. In the example given below with ”way ID” of 119882022 (and shown in Figure 3.2), the constituent nodes are shown via <nd

ref=”...”/>. For example, the first constituent node of the given way is a node with node id of 56892755 and its last constituent node is given by node id of 56892768 which is in fact the node in the previous example. Similar to a node, a way can potentially have multiple tags. For example, based on its tags, one can extract a lot of information among which are: the given way is a two-way Fox Run Parkway with four lanes, with a Primary road type. One of the tags of a way is ”maxspeed” which represents the speed limit. This tag may not always be reported for a way. A way can have similar attributes as a node such as timestamp, and uid. In addition to representing a road, a way can be also representing an area or a closed way (when the first and last nodes are the same) or other types of map polylines such as a border between two states.

```
<way id="119882022" visible="true" version="5" changeset=
"41291034" timestamp="2016-08-06T17:36:59Z" user="Caboosey"
uid="171516">
<nd ref="56892755"/>
<nd ref="56892756"/>
...
<nd ref="56892768"/>
<tag k="highway" v="primary"/>
<tag k="lanes" v="4"/>
<tag k="name" v="Fox Run Parkway"/>
<tag k="NHS" v="yes"/>
<tag k="old_ref" v="AL 37"/>
<tag k="oneway" v="no"/>
<tag k="ref" v="US 431"/>
<tag k="surface" v="asphalt"/>
<tag k="tiger:cfcc" v="A21"/>
<tag k="tiger:county" v="Lee, AL"/>
<tag k="tiger:name_base" v="Fox Run"/>
```

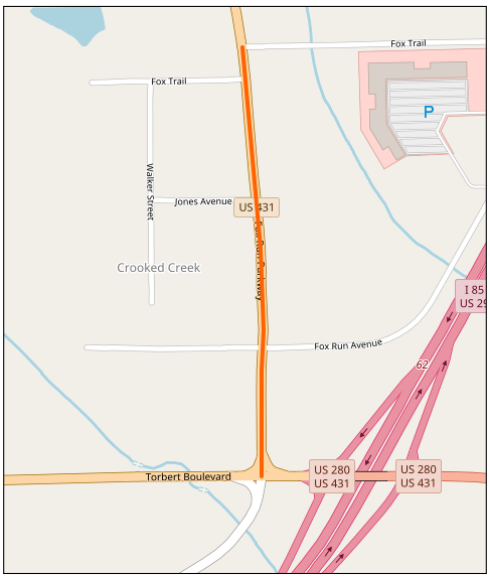

Figure 3.2: A ”way” example is shown by a solid line in color orange. A ”way” is an element type in the OpenStreetMap database.

```
<tag k="tiger:name_base_1" v="United States Highway 431"/>
<tag k="tiger:name_base_2" v="State Route 1"/>
<tag k="tiger:name_type" v="Pky"/>
<tag k="tiger:reviewed" v="no"/>
<tag k="unsigned_ref" v="SR 1"/>
</way>
```

- Relation: OpenStreetMap defines more complicated map shapes as relations. A relation can be a way with more than 2,000 nodes, or it can be areas with multiple disconnected components. More generally speaking, a relation can be composed of multiple other relations, multiple other ways and also other nodes not included in those ways. Similar to a node and a way, a relation can potentially have multiple tags. An example of a relation is given in Figure 3.3 and its OSM database information is given below. The example is a relation of id 1531549 which is a highway road (<tag k=”route” v=”road” />) with name of US 431 (Alabama). A relation can have similar attributes as a node and a way such as timestamp, and UID.

```
<relation id="2303028" visible="true" version="82" changeset=
```

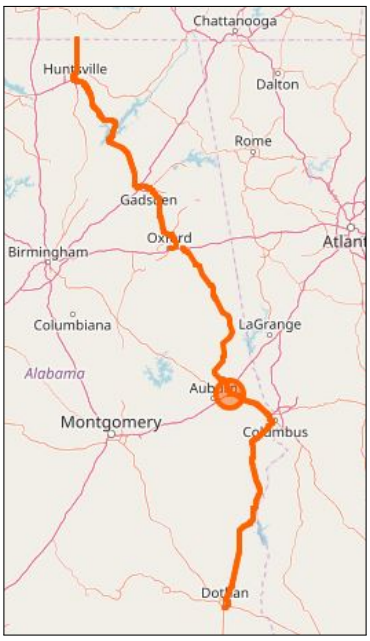

Figure 3.3: A ”relation” example is shown by a very long (more than 2,000 ”nodes”) solid line in color orange and a circle. A ”relation” is the most general element type in the OpenStreetMap database.

```
"57398534" timestamp="2018-03-21T20:14:45Z" user="micahcochran"
uid="397719">
<member type="way" ref="52210310" role="forward"/>
<member type="way" ref="224259807" role="forward"/>
...
<member type="way" ref="119882022" role=""/>
...
<member type="way" ref="128599883" role="north"/>
<tag k="is_in:state" v="AL"/>
<tag k="name" v="US 431 (AL)"/>
<tag k="network" v="US:US"/>
<tag k="ref" v="431"/>
<tag k="route" v="road"/>
<tag k="symbol" v="https://upload.wikimedia.org/wikipedia/
commons/8/81
/US_431.svg"/>
```

```
<tag k="type" v="route"/>
</relation>
```

We have downloaded the OSM database for North America. This database has more than 2.5 billion rows and a size of more than 192 gigabytes. Working with such huge database in its raw format or calling existing OpenStreetMap APIs is very slow, and therefore we need to construct a new database.

Database Construction

Based on downloaded OSM database, we have constructed three tables which are explained below.

WaysTags Table. For each way in the original OSM database, we select those ways that have the following property: way's tag "k $\in$ {"*highway*","*maxspeed*"}". If this condition is met, we add a row to the WaysTags Table with three columns of "way id," "tag key," and "tag value."

NodesTags Table. We select those nodes in the original OSM database that have the following two properties: way's tag "k = highway," and "Tag Value" is one of {"traffic signals", "giveway", "stop"}. If these two conditions are met, we add a row to the NodesTage table with two columns of "node id" and "tag value".

SingularWay Table. For every two consecutive nodes that appear in one of the ways with tag "k= highway" (which means it is a road) of the OSM database, we define a Singular Way. We collect all these singular ways in the SingularWay table. Such table has eight columns of 1. the latitude of the first node; 2. the longitude of the first node; 3. the id of the first node; 4. the latitude of the second node; 5. the longitude of the second node; 6. the id of the second node; 7. the id of the original way in the OSM database that this singular way appeared in; and 8. the order of this singular way in the original way in the OSM database that this singular way appeared in.

For example, if there is a way with id = M, in the original OSM database composed of five nodes, then there will be four singular ways defined for this specific way. The order of the first

singular way would be one, the order of the second singular way would be two, and so on. The seventh column of all these four singular ways has the value M.

We denote the new database we have constructed by

$$D = \{\rho_i^c \equiv (\rho_i^{id}, id_i^b, loc_i^b, id_i^e, loc_i^e, sl_i, t_i, ss_i^b, ss_i^e)\}_{i=1}^{N_c}, \tag{3.1}$$

which are the database of all singular ways of SingularWay table, in which the SingularWay's id is ρ_i^{id}, its type is t_i, the location of its beginning and endpoints are loc_i^b and loc_i^e, its speed limit is sl_i, and whether there is a stop location at the beginning or end of this SingularWay is shown by the variables ss_i^b and ss_i^e. Note that the speed limits sl_i^b, sl_i^e and the stop locations ss_i^b and ss_i^e do not always exist for all SingularWays; if any of them is not specified, it will be set to Null.

Data-mining Algorithm

For a given $loc_i^{Goog} = (lat_i^{Goog}, long_i^{Goog})$, the goal is to find road ρ_j defined above such that it "best" represents the road to which loc_i^{Goog} belongs. Best representation ideally means that loc_i^{Goog} is on road ρ_j. Given OSM database is very large and covers many geographical areas, in the majority of cases a road ρ_j exists such that loc_i^{Goog} is on that road. However, since OSM database is not complete, i.e., does not contain all points, it is also possible for some loc_i^{Goog} that the ideal road ρ_j does not exist in OSM database, in which case a "best" road will be chosen. We will shortly quantify the terminology "best."

For a road ρ_j in OSM database to best represent a Google Maps database point loc_i^{Goog}, the OSM road should be close enough to the Google Maps point and the angle of OSM road should be sufficiently close to the angle of vehicle's movement direction at the Google Maps point. To emphasize the importance of the angle, we provide two examples.

Example 1 - An Intersection: In the first example shown in left side of Figure 3.4, let's assume that the direction of movement is along road r_1 (south to north). In this case, since the vehicle location is closer to road r_2, road r_2 would be returned if the angle is ignored, which is incorrect.

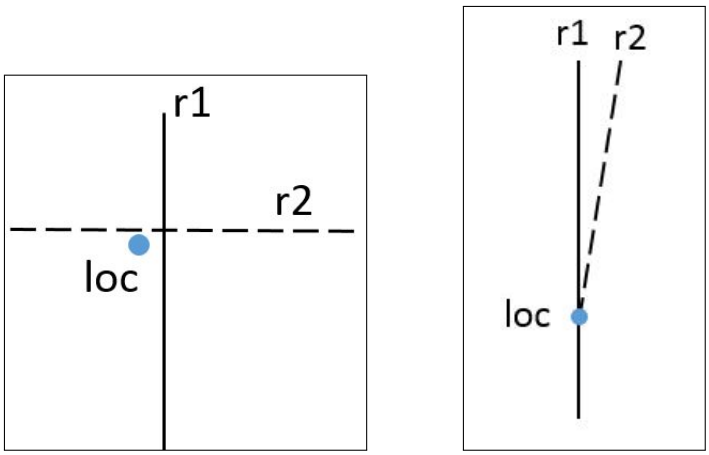

Figure 3.4: Importance of angle in road finding algorithm. If angle is disregarded, it is possible that the incorrect road r_2 is returned. Left: Intersection. Right: Highway Exit

Example 2 - A Highway Exits: In the second example shown in right side of Figure 3.4, let's assume that the direction of movement is along road r_1 (south to north). In this case, since the current location is where there is an exit from the highway, r_2 could be returned if the angle is ignored, which is incorrect.

Road Selection Algorithm

For a road $\rho_j = (\rho_j^{id}, id_j^b, loc_j^b, id_j^e, loc_j^e, sl_j, t_j, ss_j^b, ss_j^e)$ and a given GoogleMaps location of $loc_i^{Goog} = (lat_i^{Goog}, long_i^{Goog})$, let's define α_j as angle of the road, i.e., the angle of the line connecting loc_j^b to loc_j^e, and α_i^{Goog} as the angle of movement, i.e., the angle of the line connecting loc_{i-1}^{Goog} to loc_i^{Goog}. To find a road that best represents the Google point loc_i^{Goog}, we do the following. We check all roads, for each of which we consider a square area that is constructed by two points of loc_j^b and loc_j^e as its opposite corners (the blue-filled square in Figure 3.5). We then expand this area based on a specific factor of β (the larger square in Figure 3.5). If loc_i^{Goog}, is in this area, we consider road ρ_j^c as a candidate and put its id in our candidate set. We check all roads and find all candidates and put their id's in our candidate set. We call this candidate set by $S_{candidate}$. For each of the candidate roads with id j, we calculate α_j. If the absolute difference between this angle and the direction of movement (α_i^{Goog}) is less than some threshold, say $\alpha_{threshold}$ (for example $\frac{\pi}{6}$), we keep this candidate road in the candidate set of roads. Otherwise, we remove it from the set. We repeat this for all candidate roads, and if at

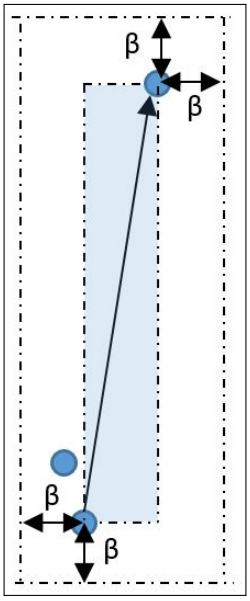

Figure 3.5: The first step of road selection algorithm (RSA): a search area (the outer square area in this figure) is constructed based on endpoints of a given route and a tuning parameter β. If the given location is within that area, the given road is selected as a candidate and passed to the second step of RSA. Otherwise, it is disregarded.

the end, the candidate set is non-empty, we select from the candidate set, the road whose angle is closest (in absolute value) to the direction of the movement. If, however, the set is empty, we go back to the original candidate set $S_{selected}$ and repeat the aforementioned filtration using $\pi - \alpha_i^{Goog}$ instead of α_i^{Goog}. The reason for the second round of filtering is that some roads are a two-way road, for which only one direction is provided in the OSM database. The second round of filtering will help us find the other direction if the one aligned with our device's movement is not found in the OSM database. If after the second round of filtering, no road is found, we return null.

The formulation of this algorithm is given below:

1. Find a selected set of roads ρ_j^c such that the following two conditions hold:

$$\begin{aligned} \min(lat(loc_j^b), lat(loc_j^e)) - \beta &\leq lat_i^{Goog} \leq \max(lat(loc_j^b), lat(loc_j^e)) + \beta \\ \min(long(loc_j^b), long(loc_j^e)) - \beta &\leq long_i^{Goog} \leq \max(long(loc_j^b), long(loc_j^e)) + \beta. \end{aligned}$$

Let's denote the set of indices of selected roads by $S_{selected}$

2. Find the solution to the following problem on the selected set of roads identified in step 1:

$$arg \min_{j \in S_{selected}} (abs(\alpha_i^{Goog} - \alpha_j))$$
$$s.t.\ abs(\alpha_i^{Goog} - \alpha_j) \leq \alpha_{threshold}$$

3. if no road is found from the previous step, run the following problem on the selected set of roads identified in step 1:

$$\min_{j \in S_{selected}} (abs(\alpha_i^{Goog} - (\pi - \alpha_j)))$$
$$s.t.\ abs(\alpha_i^{Goog} - (\pi - \alpha_j)) \leq \alpha_{threshold}$$

4. if no road is found, return null.

Speed Limit Profile

Once the road selection algorithm is finished, we do the following to get the speed limit profile S^{OSM}. Set S^{OSM} to an empty set, and $i = 0$. While $i < N_r$, do the following:

1. find the corresponding road from OSM database using the Road Selection Algorithm. If the road is null, go to step 3. If the road ρ_j^c is found, check its speed limit sl_j. If sl_j is null, go to step 2. If sl_j is not null, set: $S^{OSM} \leftarrow S^{OSM} \cup \{(m_j, sl_j)\}$ where m_j is the distance of loc_j^b from the origin of the trip, and go to step 3.

2. use the road type t_j of the road ρ_j^c found in step 1 to look up its speed limit from Table 3.1.

3. increase: $i \leftarrow (i+1)$

It is worth noting that although OpenStreetMap contains almost all routes. It is not a comprehensive source when it comes to some road information such as speed limit. We believe, though, that this database will be complete in the coming years. In light of this, when OpenStreetMap does not have the speed limit (value (v) of way tag with k="maxspeed") for a route,

Tag	**Speed limit (mi/hr)**
Motorway	70
Trunk	70
Primary	45
Secondary	45
Tertiary	45
Unclassified	25
Residential	25
Service	10
Motorway link	45
Trunk link	45
Primary link	25
Secondary link	25
Tertiary link	25
Others	5

Table 3.1: Speed Limits. The first column is the set of the road types (way tag with k="highway") in the OpenStreetMap database. The second column is the set of the speed limits we used for a road type of column one when speed limit (value (v) of the way tag with k="highway") is not specified in the OpenStreetMap data for a specific road.

we use a default speed limit based on the type of route. OpenStreetMap provides a type for a route under the way tag with k="highway". Some examples of route types are Motorway, Trunk, and Primary. For example, if the route type is primary, we use the speed limit default of 45 miles per hour.

Stop Locations

We can also extract information about locations of stop/yield signs and traffic signals from OpenStreetMap database. This information can be incorporated into our dynamic programming model. To this aim, for each stop location, we find the closest stage in the dynamic programming model to the stop location and force the speed of vehicles at these stages to be zero. Equivalently, this will decouple our dynamic programming model into multiple independent smaller dynamic programming models.

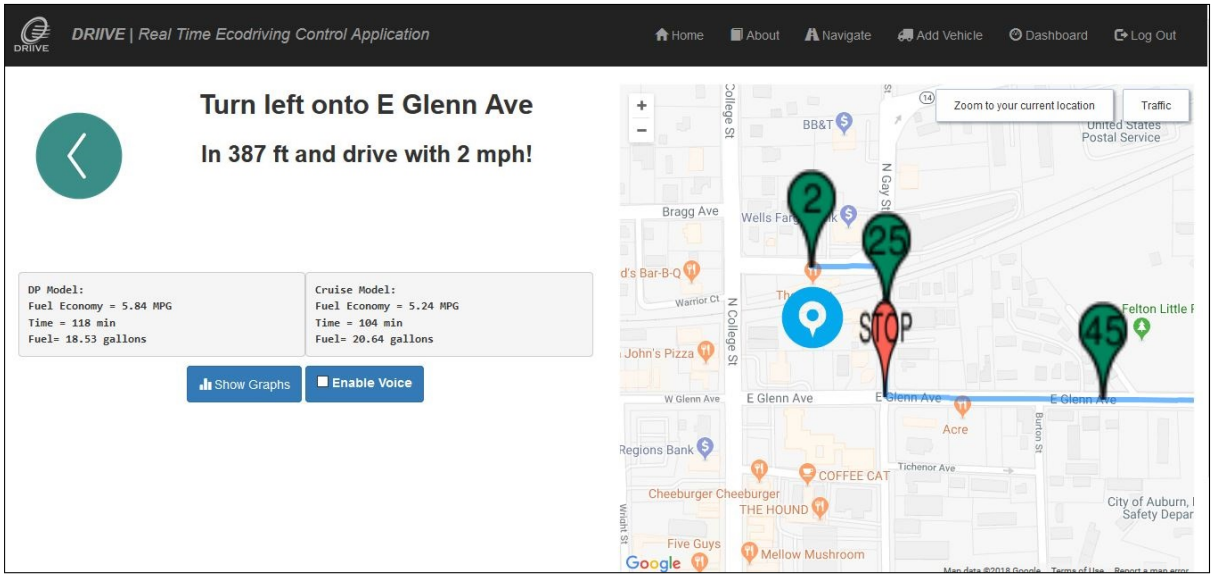

Figure 3.6: The DRIIVE page has *visual guide* (top left circle filled with color blue, currently showing an arrow to the left (West direction)), a *GPS* (an icon in color blue shown on the bottom right side of the map, and a *voice* feature (the blue left box in the middle left side of the page).

3.2.2 Weather Information

The OpenWeatherMap categorizes weather into fifty-four different conditions. Conditioned on the weather condition, we define a danger-metric with different danger levels, based on which some adjustment to the speed limit is made. At a stage of dynamic programming, the higher the danger-metric is, the higher the reduction in the speed limit of that stage will be. Based on the weather condition category, the speed limit is reduced by some percentages which are listed in Table 3.2. For example, if the speed limit range in an interstate highway when the weather condition is clear sky is $[v_{min},v_{max}]$=[50, 70], the speed limit would be adjusted to $[v_{min},v_{max}]$=[50*0.9, 70*0.9] when the weather Code is 501 ”moderate rain”.

3.2.3 GPS, Voice Activation, and Visual Guide of DRIIVE

We have developed our application such that it can use device’s GPS information to provide a navigation system with visual guides and voice activation. These new features add powerful capabilities to our applications as described in the following paragraphs. An example of these features is illustrated in Figure 3.6.

Code	Meaning	Ratio	Code	Meaning	Ratio
200	thunderstorm with light rain	0.9	201	thunderstorm with rain	0.8
202	thunderstorm with heavy rain	0.7	210	light thunderstorm	0.9
211	thunderstorm	0.7	212	heavy thunderstorm	0.5
221	ragged thunderstorm	0.5	230	thunderstorm with light drizzle	0.5
231	thunderstorm with drizzle	0.5	232	thunderstorm with heavy drizzle	0.5
300	light intensity drizzle	0.9	301	drizzle	0.8
302	heavy intensity drizzle	0.8	310	light intensity drizzle rain	0.9
311	drizzle rain	0.8	312	heavy intensity drizzle rain	0.8
313	shower rain and drizzle	0.7	314	heavy shower rain and drizzle	0.6
321	shower drizzle	0.5	500	light rain	1.0
501	moderate rain	0.9	502	heavy intensity rain	0.8
503	very heavy rain	0.7	504	extreme rain	0.6
511	freezing rain	0.6	520	light intensity shower rain	0.5
521	shower rain	0.8	522	heavy intensity shower rain	0.7
531	ragged shower rain	0.6	600	light snow	0.8
601	snow	0.7	602	heavy snow	0.5
611	sleet	0.5	612	shower sleet	0.5
615	light rain and snow	0.8	616	rain and snow	0.8
620	light shower snow	0.8	621	shower snow	0.5
622	heavy shower snow	0.5	701	mist	0.9
711	smoke	0.9	721	haze	0.8
731	sand, dust whirls	0.6	741	fog	0.6
751	sand	0.8	761	dust	1.0
762	volcanic ash	0.8	771	squalls	0.8
781	tornado	0.5	800	clear sky	1.0
801	few clouds	1.0	802	scattered clouds	1.0
803	broken clouds	1.0	804	overcast clouds	1.0

Table 3.2: Weather conditions. The first and second columns are respectively the code and description of different weather conditions in the OpenWeatherMap condition database. The third column is the multiplicative factor by which a speed limit is adjusted for a specific weather code.

GPS

The app receives the current location of the user using the device's GPS. It will show the direction, the speed that the driver should start with, and the distance from user's current location to the next place at which user need to make an action such as exiting a highway. Similar to other navigation applications, DRIIVE app also updates the direction guide (distance and direction description) according to vehicle's location as the vehicle moves. DRIIVE also observes the user's location in order to update the dynamic programming proposed velocity. In other words,

Priority	Keyword	Direction
1	Turn left	West
2	Turn right	East
3	Keep left	Northwest
4	Keep right	Northeast
5	Left	West
6	Right	East
7	Head	North
8	Continue	North
9	Take exit	Northeast
10	Merge	North

Table 3.3: Visual Direction. Keywords are given priority for look-up.

the proposed driving speed will change as soon as the vehicle passes the road portion associated with that speed.

Visual Guide

Further, we have enabled our app with this feature that the directions such as turning left or right are also shown visually on the app for the driver to understand the direction easier. Google does not provide direction icons and only provides the direction instruction in text format. We need to write an algorithm that reads the text and decide on the direction. To this aim, we have given some priority to the keywords shown in Table 3.3 and searched for these keywords in priority. For example, we will search for the keyword ”Turn Left” first and then search for keyword ”Merge” because ”turn left” has a higher priority compared to ”merge”. This prioritized-search algorithm is required. Otherwise, there are cases that would result in a wrong direction with a normal (un-prioritized) search. Once the proper keyword(s) is found, we check if it is different from the one found in the earlier step. If so, we display this extracted direction visually on our app.

DRIIVE also uses Google Maps features to show user's location, the direction, and the desired velocity on the map. The velocity will be shown to the user as a number in miles per hour inside a green marker at the exact location on the route. If the user needs to decelerate, the marker becomes red.

Voice Activation

Getting distracted by devices such as smartphones increases the chance of road accidents. Therefore, we have enabled our app with a voice such that a driver does not need to look at the app every time to get the information from our app.

3.2.4 Enhancing DRIIVE

The web app will allow the users to specify the following parameters on the web:

- Fuel consumption importance level. This parameter indicates how much fuel consumption is important to a driver relative to the overall trip time. The higher this parameter is, the more eco-friendly the resulting speed profile will be, and as a trade-off, the more time consuming the overall trip will be. The fuel consumption and the overall trip time importance levels are respectively denoted by w_1 and w_2 in 2.16.

- The action level. This parameter will determine how much driving comfort level a driver expects. The comfort level has a reverse relation with the number of times that the driver has to change vehicle speed and brake. The action levels corresponding to the amount of change in vehicle speed and gear are respectively denoted by w_3 and w_4 in 2.16.

- Acceleration limits. The app allows a user to enter their desired maximum and minimum acceleration limits. This is a very useful feature that the app provides, given the fact that different users have different tolerance of acceleration at different times and on the different situations. For instance, in snowy weather, a driver prefers to change its speed at a slower pace and therefore can put in the app smaller values for max and min acceleration compared to sunny weather. Or, another example is when a baby is on board, in which case slower change of vehicle speed is more desired. Or, another example is for trucks when their loads include hazardous and heavy material, in which case a smaller range for allowed acceleration is safer.

- Vehicle Mass. Users should specify the weight of their vehicles. This information is being used in the dynamic programming model. As our analysis show, and is expected, the results of the web application highly depend on the weight of the vehicle.

- Vehicle-related data. Users are able to specify their vehicles by entering their vehicle-related data into the application before running the application. This assures that the speed profile optimization is performed on users' specific vehicle. These vehicle-related data are described in Section 2.2.5.

3.3 Web Application

In this section, our web application is described from a software engineering design perspective. The front page of this web application is implemented in HTML, CSS, and JavaScript. Its back-end is implemented in Python and django. MySQL [22] is used for database construction and making queries from those databases.

To be able to use this web application, users need a device that has GPS, Internet connection (to collect real-time information), and a modern browser that supports HTML, CSS, and JavaScript.

For this web application to be adaptable in screens of various sizes, we have used Bootstrap libraries. We have only shown two design sizes of extra-small (smartphone) and medium (tablets) in this section.

This web application has nine main web-pages that are described in details in the rest of this section.

3.3.1 Home Page

The first page of our web application needs to be simple and to provide the user access (links) to five other pages of "About", "Navigate", "Add Vehicles", "Dashboard", and "Log-in." Figure 3.7 shows two wireframes for the page "Home"; one for small-size devices such as smartphones, and one for medium-to-large size devices such as tablets, laptops, and PCs. Figure 3.8 shows the final version of "Home" page.

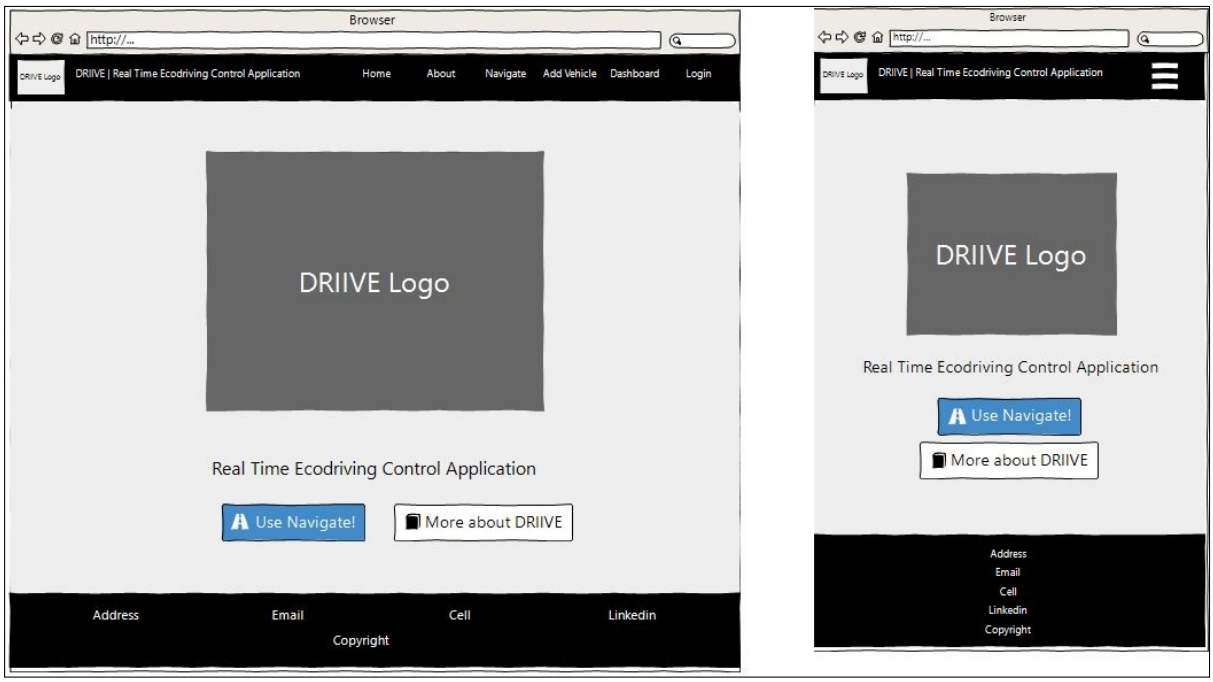

Figure 3.7: Wireframe for page ”Home”

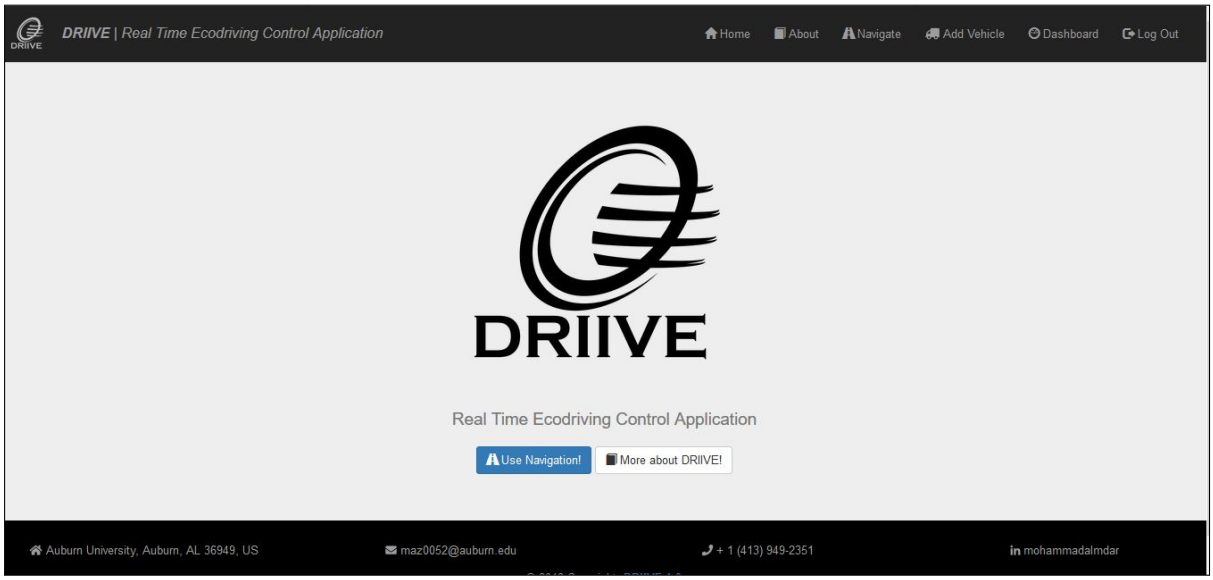

Figure 3.8: A snapshot of the final version of page ”Home.”

3.3.2 About Page

This page should aim to provide more information about the project and should not be log-in required. The first part of this page should describe the project's motivation and aim. The second part should provide a short video about this web application, its features and act as a step by step guide on how to use each page and the software.

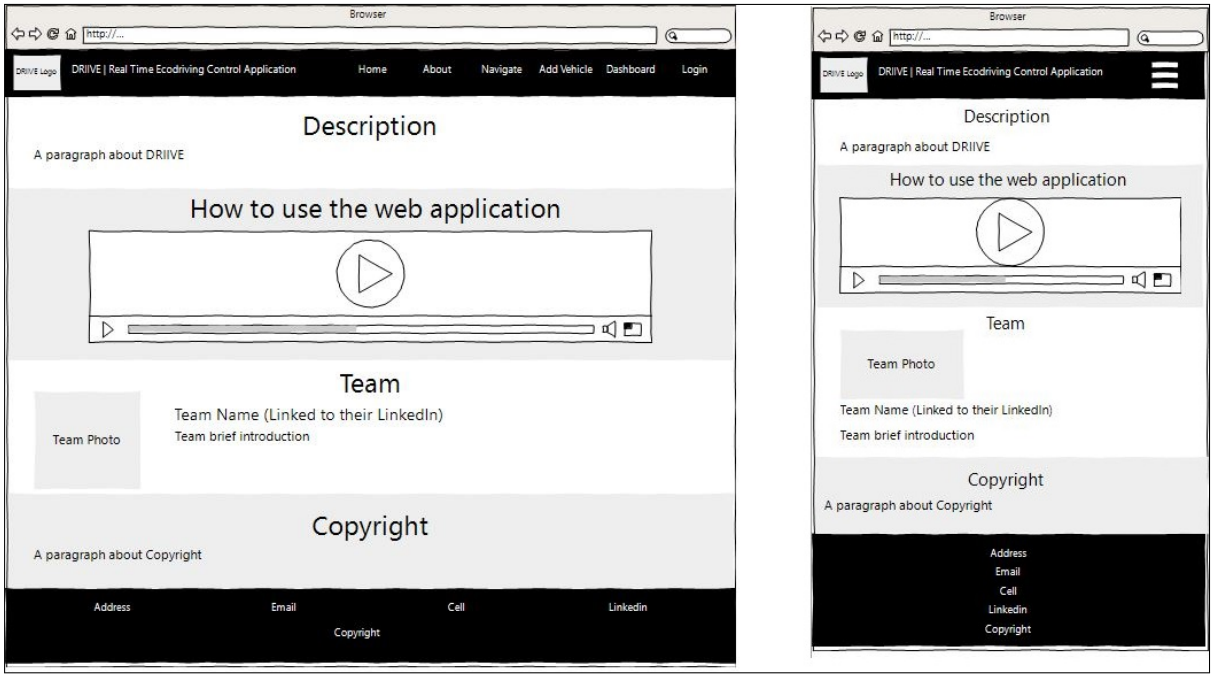

Figure 3.9: Wireframe for the page ”About”

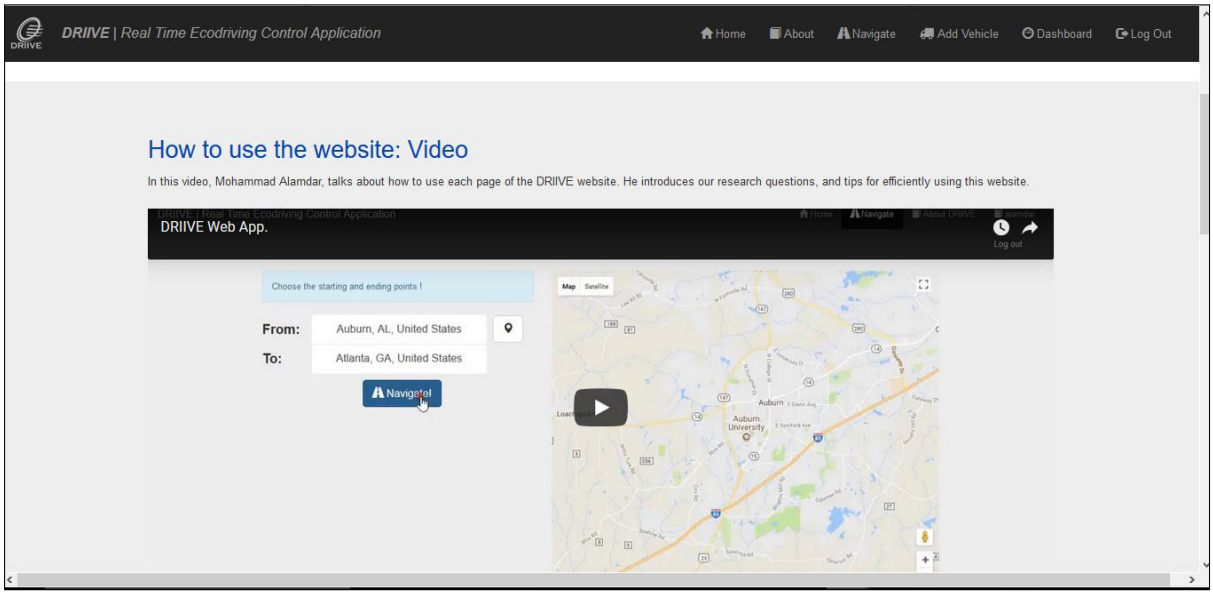

Figure 3.10: A snapshot of the final version of the page ”About”.

Figures 3.9 and 3.10 show wireframes and a snapshot of the final version for the page ”About”, respectively.

3.3.3 Log-in Page

Some of the web application pages require a log-in. Therefore, the ”log-in” page must be able to allow for a new user to register, and also allow for already-registered users to log in. The

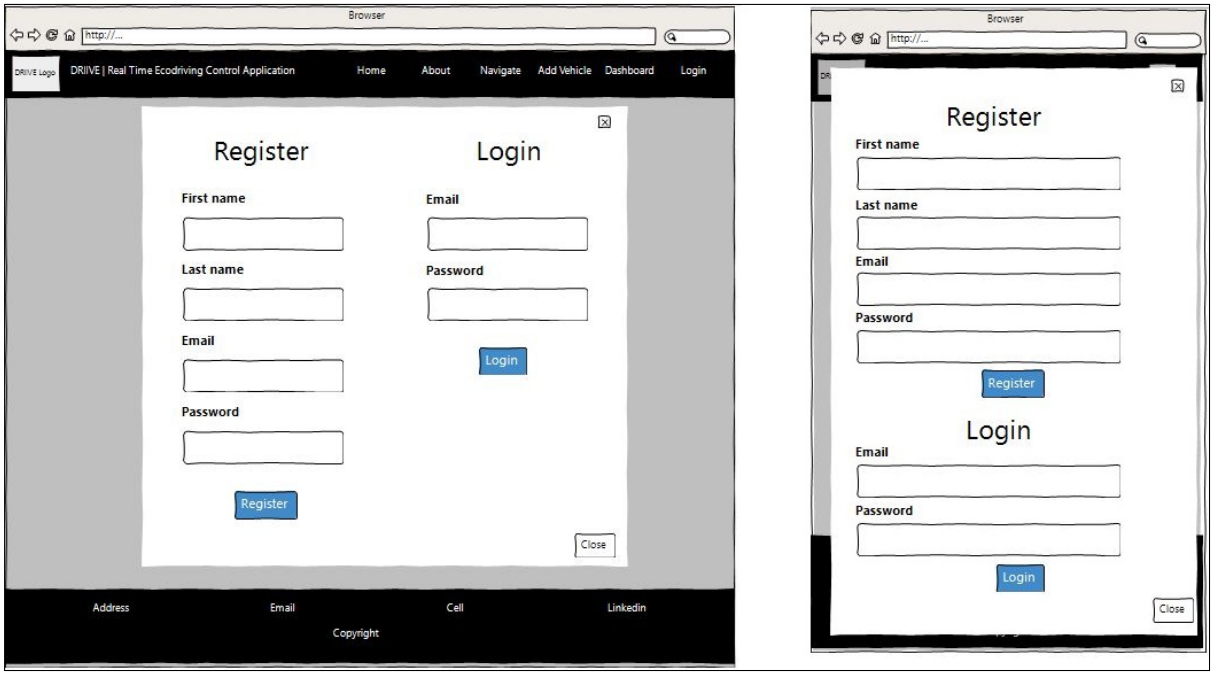

Figure 3.11: Wireframe for the page ”Log-in”

required information for registration is a name, a password, and an email address. The design of this page should be such that if the required fields for registration are not filled or are incorrectly filled, a relevant warning message pops up and asks the user to check the fields again.

Figure 3.11 shows the wireframe for the page ”log-in”.

Users’ information is stored in a safe database on the server. When the web application requires this information, it connects to this database and uses Python to read this information from the server. A snapshot of the final version for ”Log-in” page is shown in Figure 3.12.

3.3.4 Add Vehicles Page

This page must require a log-in first and should aim to allow users to specify their own vehicle by providing detailed information of their vehicle and saving them for future use. Users must be able to edit their already-added vehicles at any time. Such information should contain brand, model, year, and mechanical characteristics of the vehicle. The design of this page must be such that it provides a user with the name of the information along with its unit such that there is no ambiguity or confusion for the user. The page must provide some default values and units for the required information, and allow users to change them. Further, a link to a reference must be

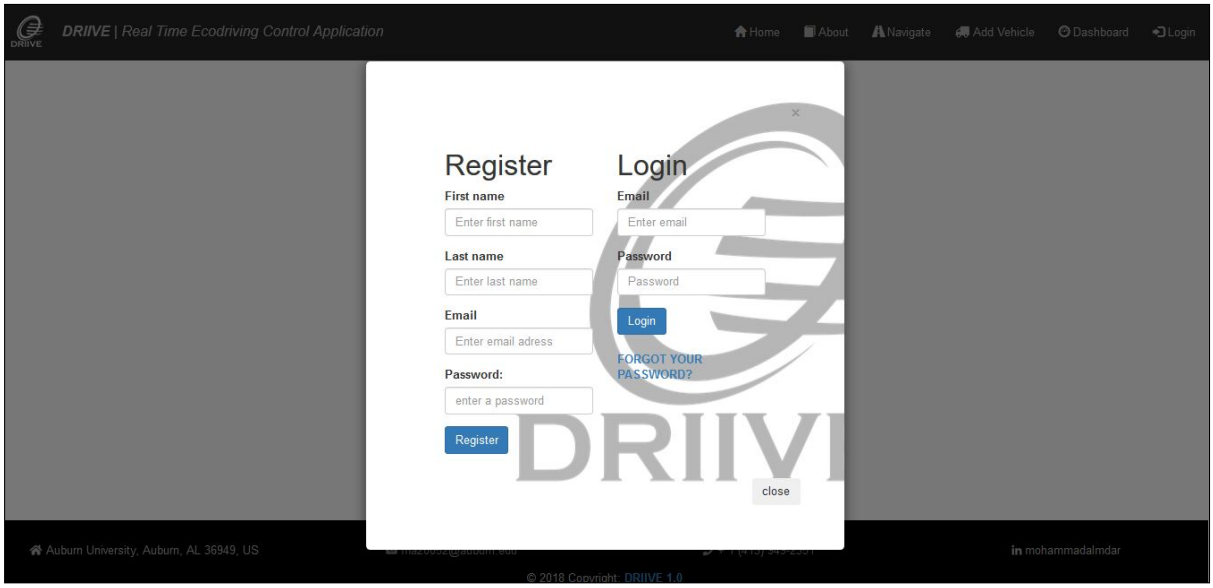

Figure 3.12: A snapshot of the final version of the page ”Log-in”.

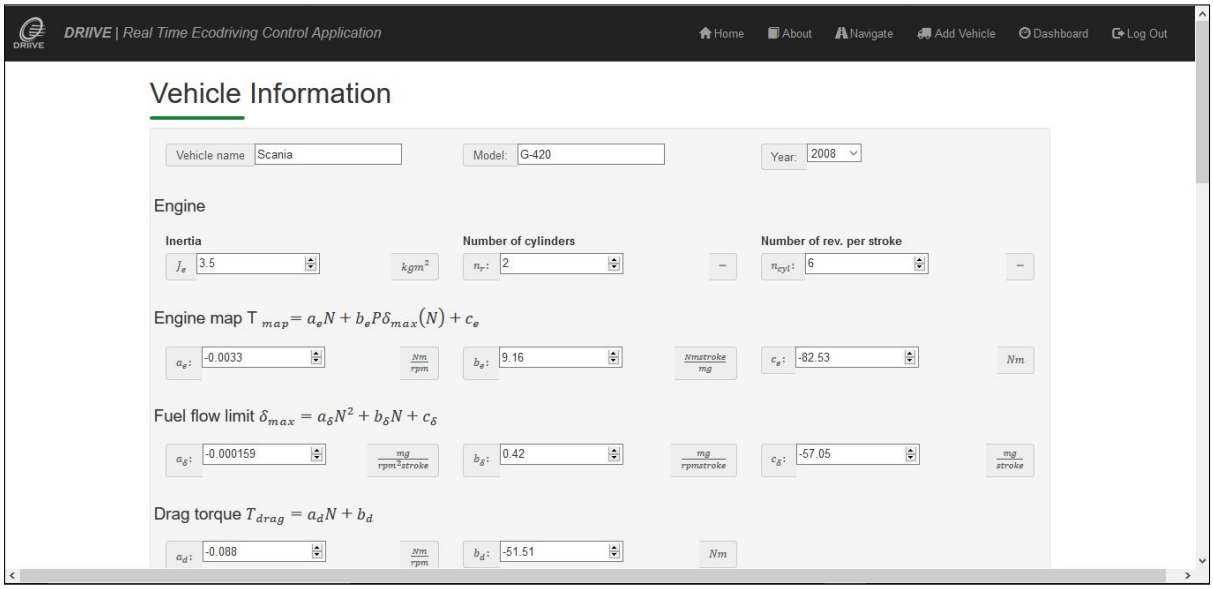

Figure 3.13: A snapshot of the final version of the page ”Add Vehicles”.

provided in which all this information is explained. The page must also provide two options of ”Apply the changes” and ”Cancel” in order to allow for a user to submit the changes or cancel a specified vehicle.

3.3.5 Dashboard Page

This page must provide users with the following features:

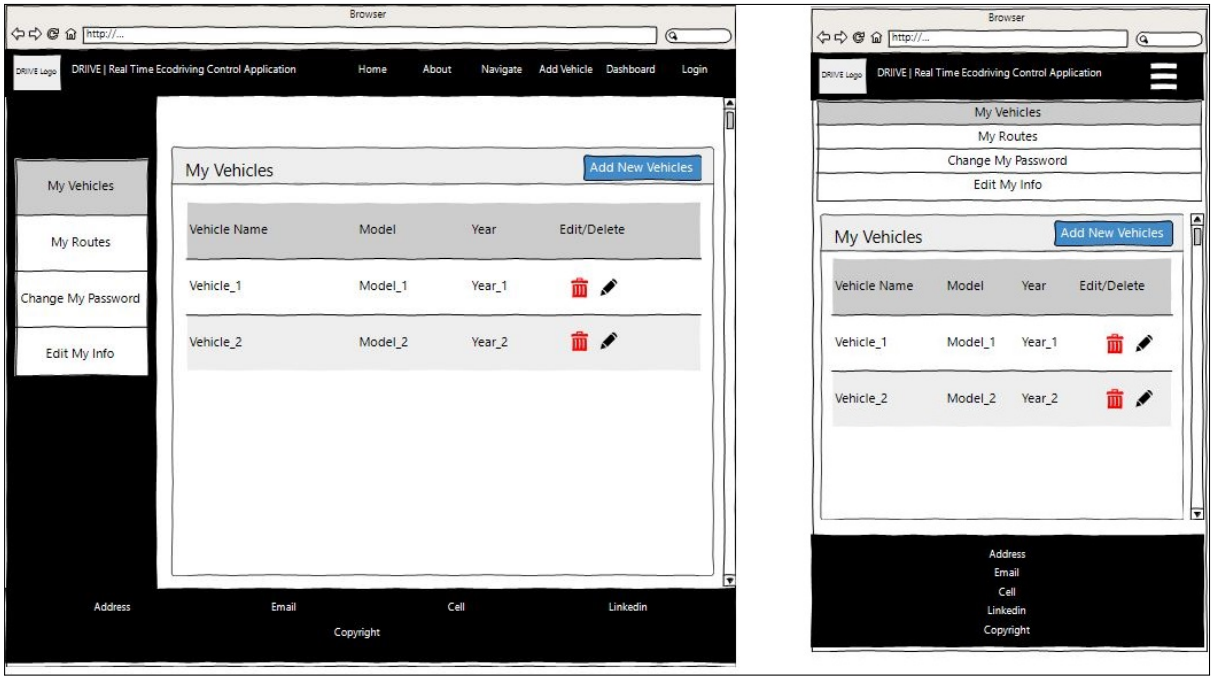

Figure 3.14: Wireframe for the page ”My Vehicles”

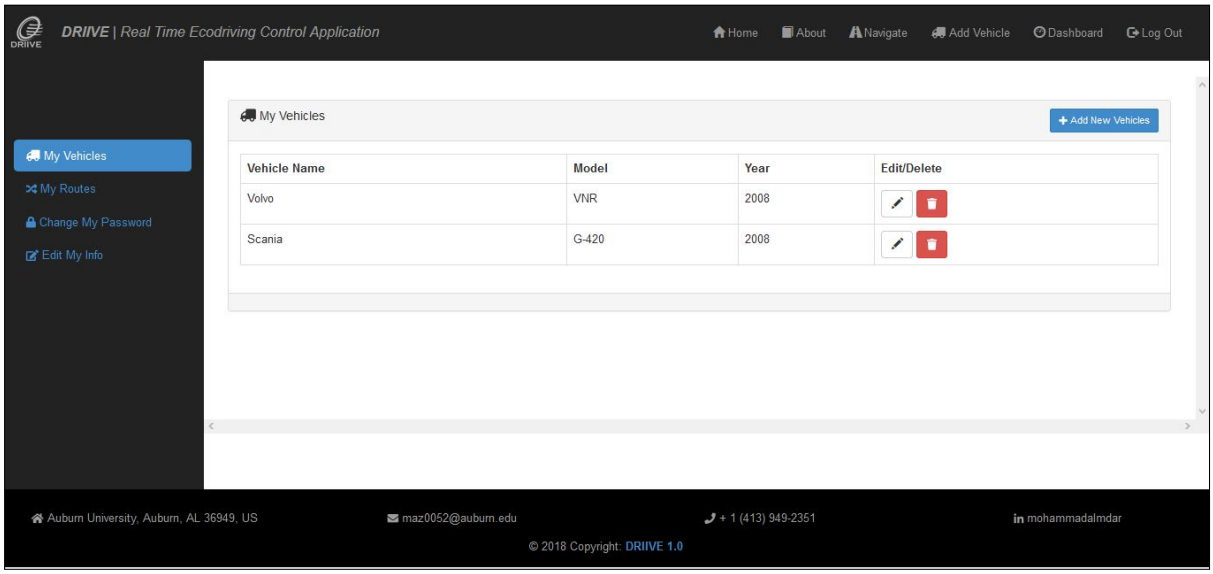

Figure 3.15: A snapshot of the final version of the page ”My Vehicles”.

- My Vehicles. This tab must enable users to see all their vehicles, to edit or delete the existing vehicles. There must also be a button that allows users to add new vehicles. This must be done by linking this page to the ”Add vehicle” page described above through this button.

  Figures 3.14 and 3.15 show the wireframe and a snapshot of the final version for this tab, respectively.

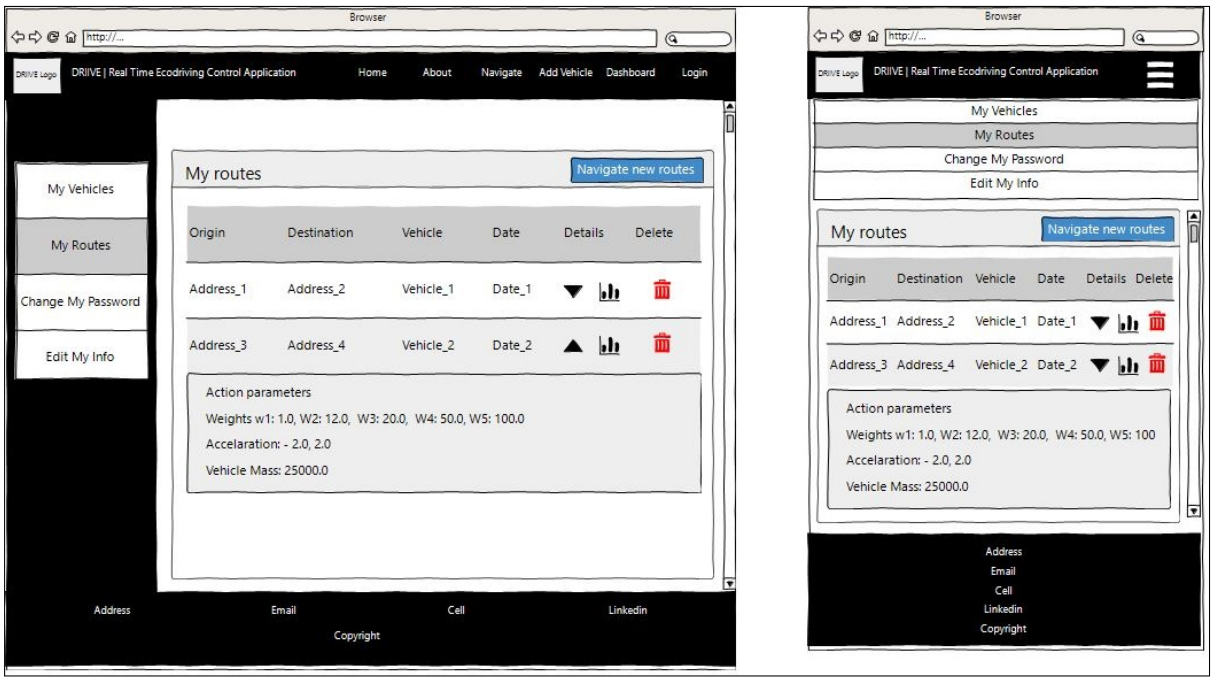

Figure 3.16: Wireframe for the page ”My Routes”

- My Routes. Each time a user uses this web application, the data related to the use must be recorded. This tab must display a summary of all stored records related to the previous uses by this user. The records must contain addresses of origin and destination, the vehicle, date, and time.

  There must also be a feature on this page for a user to see the details of each record, such as dynamic programming parameters and the vehicle mass set by the user.

  Further, there must be a link provided for each record that connects to another page ”Drive” page which displays all outputs related to that record such as the output fuel economy performance, the set of speed, and direction guidelines. This ”Drive” page features must be there for reusability convenience.

  Figures 3.16 and 3.17 show the wireframe and a snapshot of the final version for the tab ”My Routes”, respectively.

- Edit My Info. Users should also be able to edit their personal information such as name and the email address at any time.

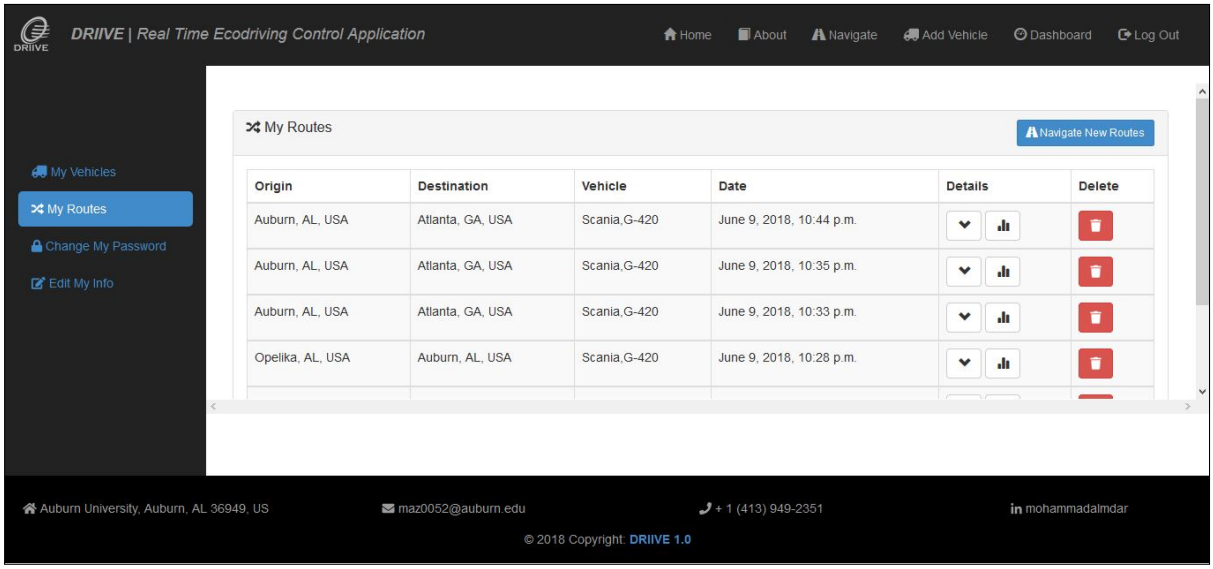

Figure 3.17: A snapshot of the final version of the page ”My Routes”.

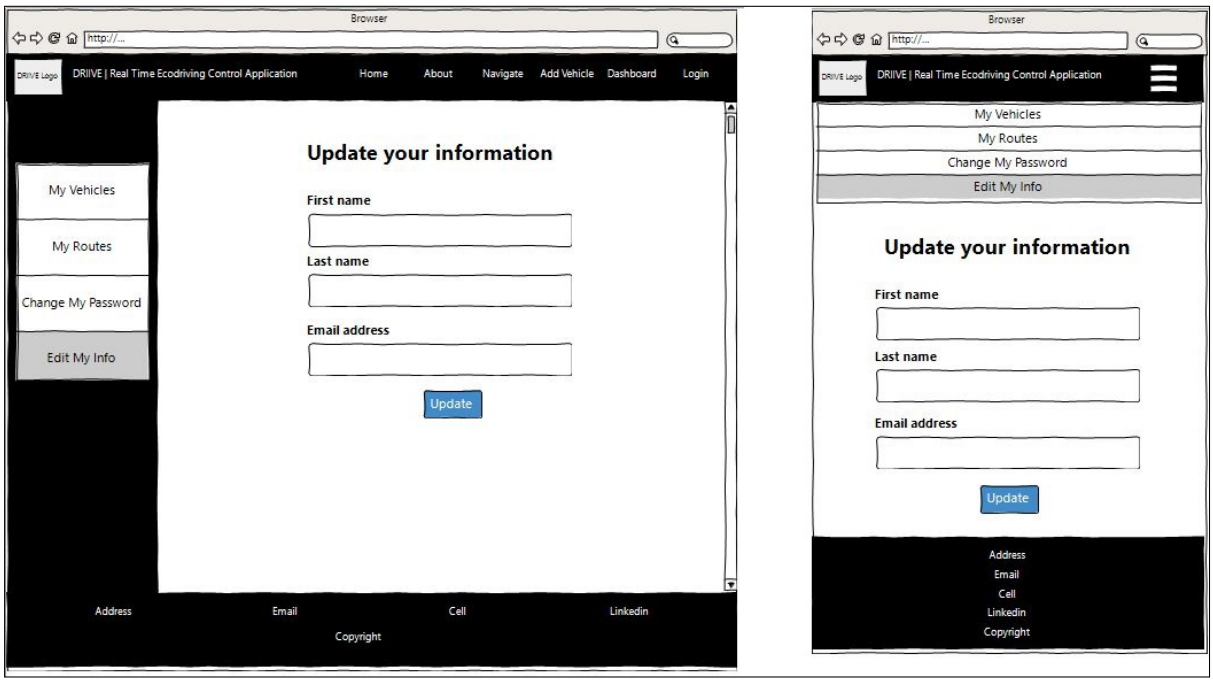

Figure 3.18: Wireframe for the page ”Edit My Info”

Figures 3.18 and 3.19 show the wireframe and a snapshot of the final version for this tab, respectively.

- Change Password. Users should also be able to update their password at any time. It must ask users for their old password, and new passwords twice.

  Figures 3.20 and 3.21 show the wireframe and a snapshot of the final version for ”change password” tab, respectively.

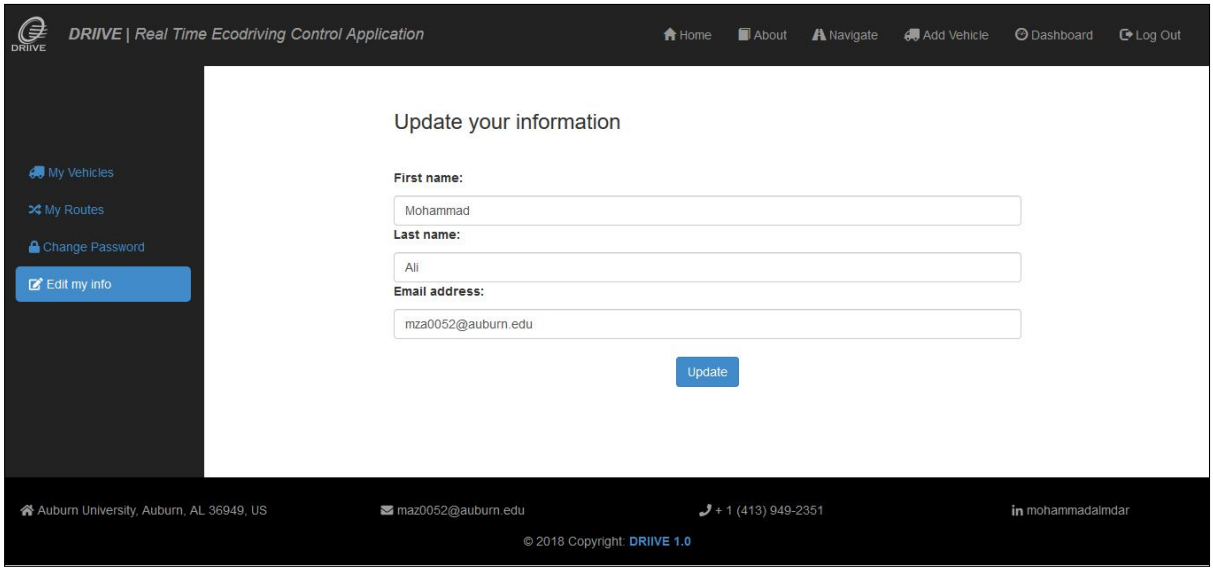

Figure 3.19: A snapshot of the final version of the page ”Edit My Info”.

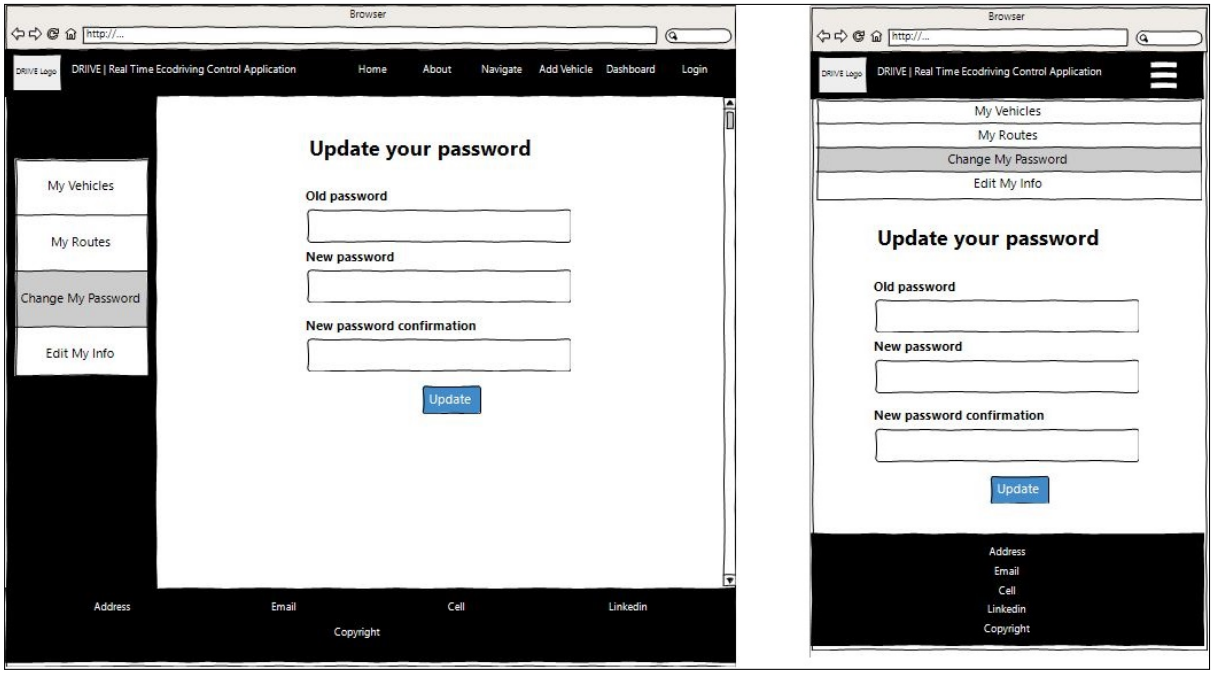

Figure 3.20: Wireframe for the page ”Change Password”

Dashboard Data Model

To implement the ”Dashboard” page, we have used the web framework of django [23] to design and implement our database. We have created three SQLite data tables of Users, Vehicles, and Routes (3.22).

The ”Users” table has five fields of ”user id”, ”first name”, ”last name”, ”email” and ”password”. This data table stores users information. A row is added to this database once a

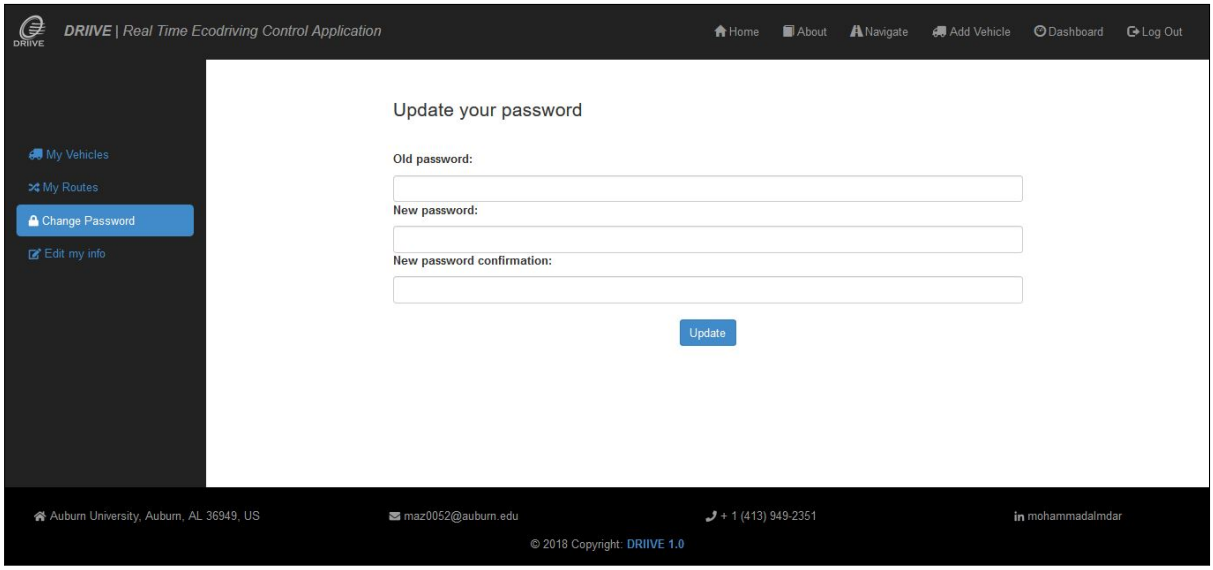

Figure 3.21: A snapshot of the final version of the page ”Change Password”.

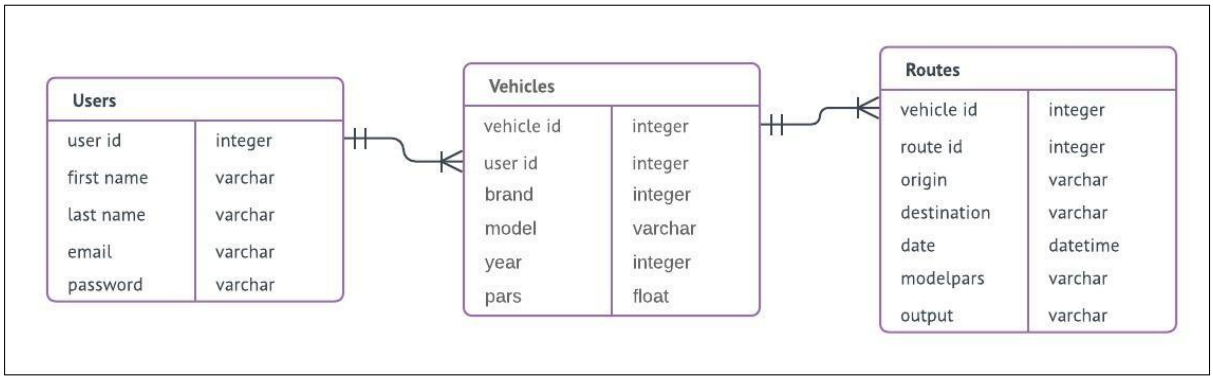

Figure 3.22: Three SQLite tables of ”Dashboard” database and their fields.

new user registers in the ”Log-in” page of our web application. Also, once an existing user logs into the web application, their information is compared against the information of this table. In tabs ”Update My Info” and ”Change My Password”, the fields of this table gets updated.

The ”Vehicles” table has six fields of ”vehicle id”, ”user id”, ”brand”, ”year”, and ”pars”. This data table stores vehicles information. The field ”pars” is a set of parameters related to the vehicle. A one-to-many relationship exists between the two tables of ”Users” and ”Vehicles”, which means one user can have many vehicles and a vehicle can only be in one user. The data of this table gets added, edited or removed from the page ”Add Vehicles” and from tab ”My Vehicles” of page ”Dashboard”.

The ”Routes” table has seven fields of ”vehicle id”, ”route id”, ”origin”, ”destination”, ”date”, ”modelpars” and ”output”. This data table stores all the outputs of the optimization.

The ”output” field contains all the outputs of dynamic programming and cruise controller for up to three routes. The ”modelpars” are dynamic programming parameters that the user has entered. A one-to-many relationship exists between the two tables of ”Vehicles” and ”Routes”, which means one vehicle can be used for many routes and a route can only be in one vehicle. Once a vehicle is deleted, all routes related to that vehicles are also removed from the ”Routes” data table.

In tab ”My Routes” of page ”Dashboard”, all data related to the users will be pulled from the database. A user can delete a route from the database as wells as looking at the summary of the route. If the user wants to see more details of the route, the application will transfer the output column of the route to the page DRIIVE.

3.3.6 Navigate Page

This page must provide two input fields in order for the user to enter the trip’s origin and destination addresses. A button must be provided on this page that collects current location of the user. There must also be a button on this page that once clicked will use the entered addresses and collects from Google Maps database the information of up to three routes between the two addresses. An interactive map must be displayed on this page that shows all the routes for the user to be able to verify the addresses and routes visually . Further, the page must provide a warning box in which current ”status” and ”next step” are shown as a guideline. This box should be visually color-coded; if the box is giving some information, it should be in color blue; if there is an error message, it is visually better for it to be in color red; if there is a no-error message, a green color is more suited.

Once a route is selected, the page must provide to the user all the vehicles the user has previously specified (for the user to select from among them). Next to these already specified vehicles, there must be an ”Add” button that once clicked takes the user to the ”Add Vehicles” page for the user to specify and add a new vehicle, if desired. At this step, the page must provide to the user, names of the parameters that are used in the dynamic programming model with some default values, and asks the user to change their values. These parameters are minimum acceleration, maximum acceleration, and coefficients of the objective function variables which

are fuel, time, velocity change and brake. Vehicle mass can vary by time (depending on the vehicle's load), users must be able to enter their vehicle mass at this page. For all these fields to be displayed at once, the page must provide a user with a button that hides or unhides the original and the destination addresses.

Once all info is entered, there must be a "Next" button on the page for the application web to run on this information and go to the "Summary" page. For the application to run, it must connect to specific databases to collect some information from them. After all these data are collected, a dynamic programming runs. An animation (or a loader GIF) must pop up to inform the user when a database is connected to or when a dynamic programming is running in the order itemized below.

- Connecting to Google Maps API to collect elevation information from the Google Maps database. A progress bar must be displaying the percentage of the collected information.
- Connecting to the OpenStreetMap API to collect road information.
- Connecting to the OpenWeatherMap API to collect weather conditions.
- Solving dynamic programming

Once each step above is done, a text message must appear to inform the user.

"Navigate" Front and back-ends Interaction

In this section, we describe how the front and back-ends of page "Navigate" interact with each other and with the rest of the application (Figure 3.23). For the front end of page "Navigate" to connect to Google Maps API's (Autocomplete API, Direction API and Elevation API) and collect route and elevation data from them, we have written codes in JavaScript that takes the users' inputs for the origin and destination addresses and returns location, route directions and elevations.

"Navigate" front end then sends routes, vehicle id, dynamic programming parameters which were selected by users directions, and elevations to the back-end of the page "Navigate". The back-end of page "Navigate" is composed of three main components. The main

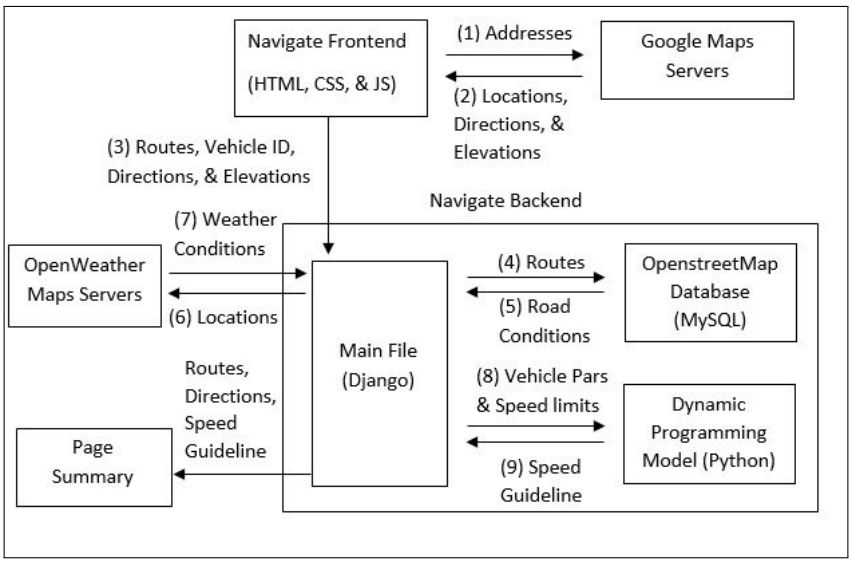

Figure 3.23: Navigate Front and back-ends. A schema of the front and back-ends of page "Navigate" and how they interact with each other and with the rest of the web application.

component is a Python Main File in django that manages the back-end functionalities. We have downloaded the OpenStreetMap data for North America in an XML file. Using some lines of code in Python, each row of this XML file is read and the desired database (of Section 3.2.1) is constructed in MySQL. Django connects to this database to read road conditions. The main file then connects to the OpenWeatherMap API and collects the weather conditions of the route. The main file then uses road and weather conditions to find speed limits of all portions of the route. It then uses vehicle id (from the front page input set) to get vehicle's parameters from the "Vehicle" table, and pushes all this information to the dynamic programming model. This dynamic programming model is a separate file on the server written in Python and located in the back-end. The outputs of the dynamic programming are the optimal speed profile, the overall trip time, and average fuel consumption. To have a benchmark for comparison, a cruise controller model under the assumption of driving with the speed limits will be run. cruise control model will return the same set of output as dynamic programming. Once the main file has speed profile for both models, directions, and routes, it pushes them all to the next page "Summary".

Figures 3.24 shows a snapshot of the final version of the page "Navigate". Figures 3.25 shows the same page when the user is waiting for the dynamic programming model to solve the optimization.

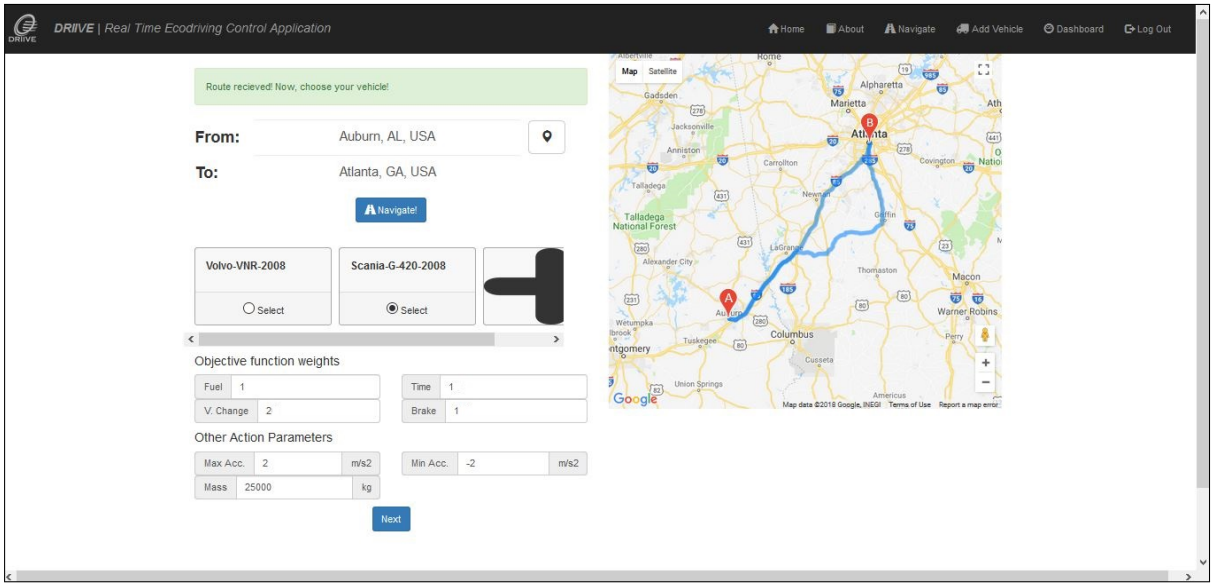

Figure 3.24: A snapshot of the final version of the page ”Navigate”.

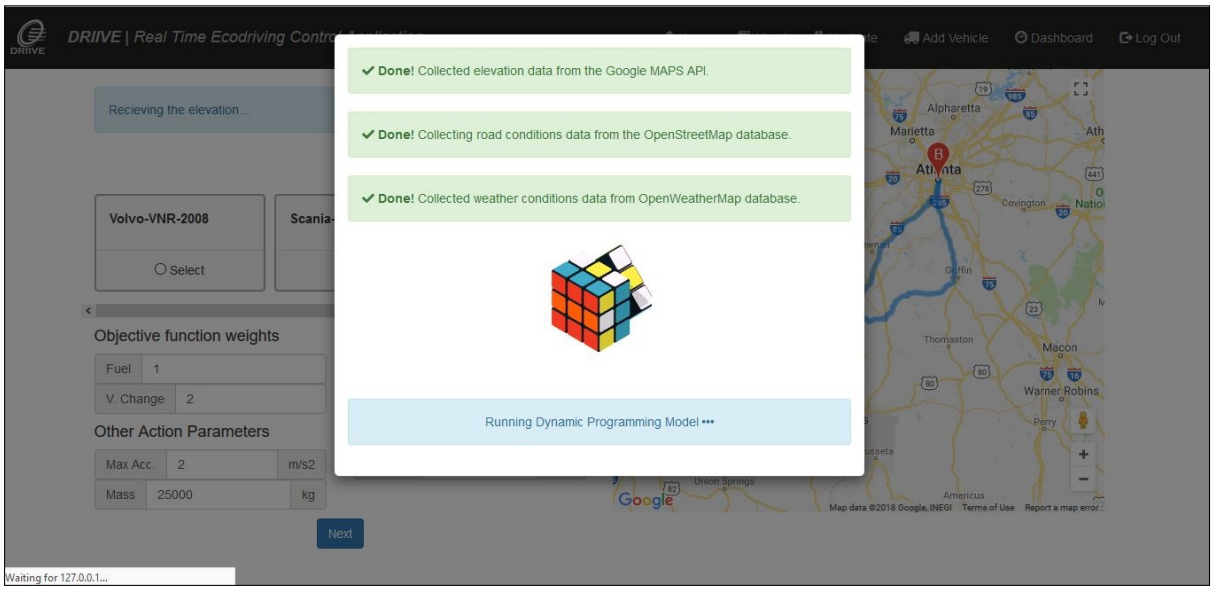

Figure 3.25: A snapshot of the final version of the loader animations in the page ”Navigate.”

3.3.7 Summary Page

The ”Summary” page must show a summary of the results of dynamic programming for each route. The summary must identify the route, the time of the trip (in minutes), the length of that route (in miles), the fuel consumption (in gallon), and its rate (miles per gallon) of the trip on that route. The benefit of having the summary of all trips collected in one place on this page is that users will then make an educated selection from among the given routes based on the

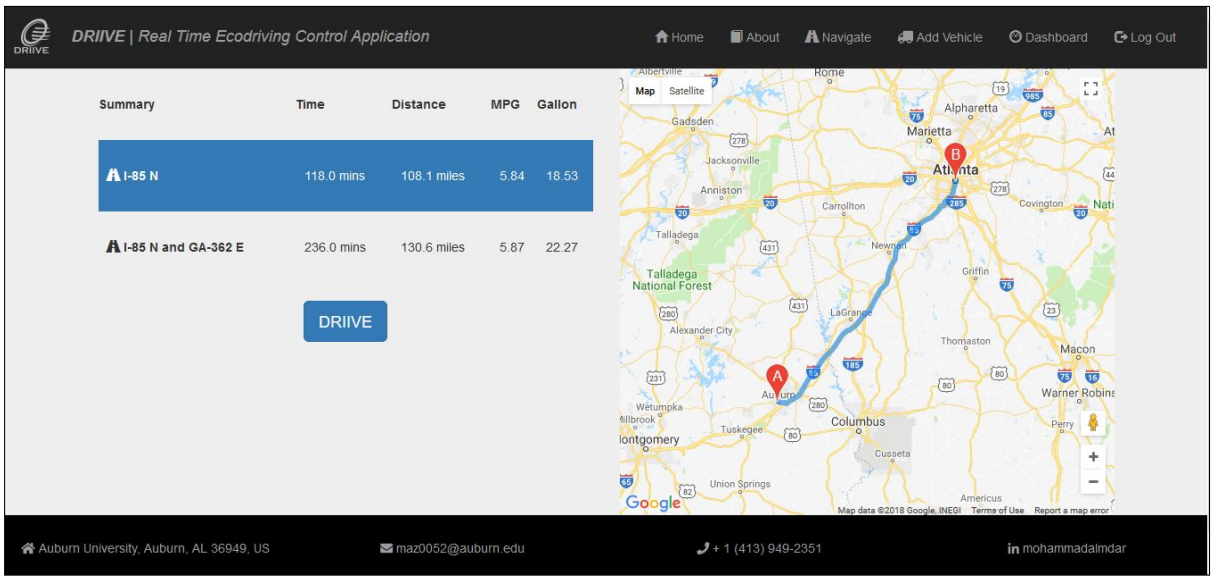

Figure 3.26: A snapshot of the final version of the page ”Summary”.

provided summary. There must also be a map provided on the page for the user to visualize each route.

The page must provide a ”DRIIVE” button on this page for users to get the speed profile of the route they select. Once button ”DRIIVE” is clicked, the user is taken to the ”DRIIVE” page. Figures 3.26 shows a snapshot of the final version for the page ”Summary”.

3.3.8 DRIIVE Page

Once the optimal solution of the dynamic programming for each route is found (in Navigate page) and the user has selected the route (Summary page), the optimal speed profile of the selected route is returned to the DRIIVE page. The speed profile specifies the speed that the driver should use every 0.1 miles such that the overall fuel consumption is reduced.

In the ”DRIIVE” page the following must be shown:

- Directions and instructions of the movement, the speed on which the vehicle must drive, and
- expected fuel consumption (in gallons), fuel consumption rate in miles per gallon, and trip’s time for dynamic programming and cruise control.

This page must have voice enabled by default. The voice tells the driver the directions. There must be a button on this page that allows the user to disable this voice feature.

There must also be an interactive map that displays the route and the speed profile of this route at the vehicle's current location. User's current location must also be shown on the map by an icon so that the icon moves as the vehicle moves on the route. The map must be kept zoomed-in around the current location icon. A button must be added to let users disable a zooming-in feature in the case that users want to see the larger part of the route. The direction and speed guide must also be updated as the vehicle moves on the route.

Another button must also be provided on this page that, once clicked, displays the traffic layer on the map provided by Google Maps database. It helps drivers have a better sense about the traffic speed of the route. The Google Maps traffic layer displays traffic speed with four colors of green as the fastest traffic, orange, light red, and dark red as the slowest traffic.

In addition, a "Show Graphs" button must exist on this page that, once clicked, will take the user to the page "Graphs".

All codes on this page are written in JavaScript. We have used responsiveVoice.js [24] library, the Google Maps JavaScript API and the HTML DOM Navigator Geolocation property for voice feature of our web application, to show the location of the user and the route on the map, and to get user's current location every second, respectively.

DRIIVE page pseudocode

The pseudocode of the algorithm used for this page is given below:

1. Get current location from HTML DOM Navigator Geolocation property.

2. Set the boolean variable isNewStep to true.

3. Read the next step from Route Output and store the coordinates of the next step in variable nextLoc

4. Read the Direction text to nextLoc from Routes Output and update the Direction text field in HTML.

5. Analyze the direction text to find the correct direction degree and rotate the direction circle

6. Show the current location on Map

7. Get the distance between current location and nextLoc by calling Google Map Geometry API and update the Distance field in HTML.

8. Read the speed Guide from Dynamic Programming Output and update the speed field in HTML

9. If isNewStep= true call responsivevoive.js to speak the direction box for the user

10. Set isNewStep to false

11. Get current location from DOM Navigator after a second

12. Get the distance between current location and nextLoc by calling Google Map Geometry API.

13. If user reached the nextLoc

14. If nextLoc is the destination, update the direction field with You have reached your destination and STOP, else go to STEP 2.

Figures 3.27 shows a snapshot of the final version of the page ”DRIIVE” when a route is selected from Auburn, AL to Atlanta, AL.

3.3.9 Graphs Page

For each route, the output of the dynamic programming must be shown in multiple diagrams in the page Graphs of our application.

In the first set of interactive figures of this page, users must be able to select up to six possible routes (a dynamic programming and a cruise control for each route) for comparison purposes. Seven diagrams must be displayed to show velocity, throttle, gear, brake, acceleration, elevation, and distance from the origin for each of selected routes. The x-axis represents the

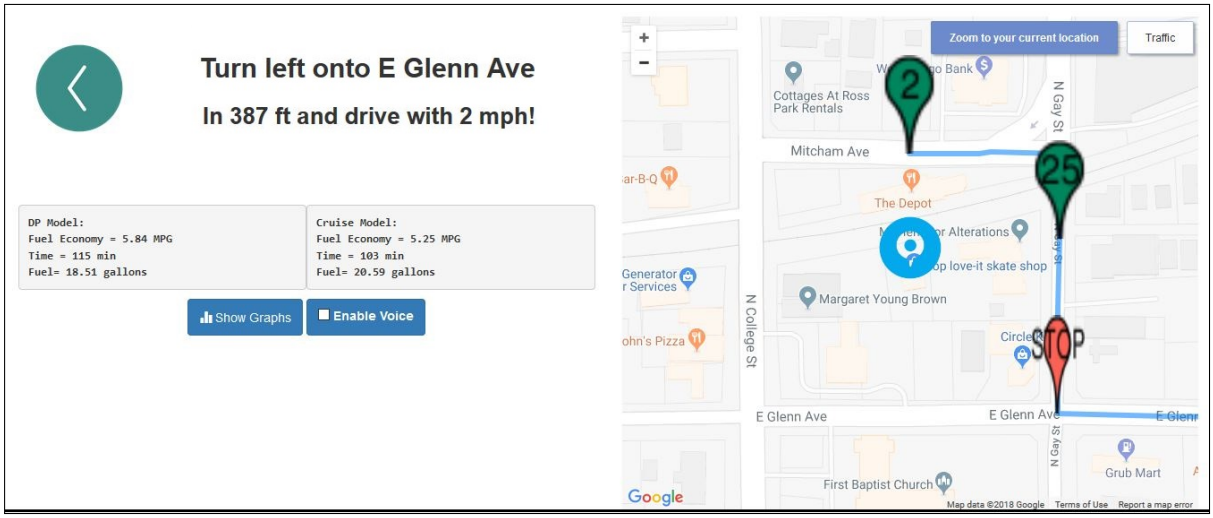

Figure 3.27: A snapshot of the final version of the page ”DRIIVE”.

time of the trip. Further, when a cursor hovers over a location, all the values of seven diagrams for that time must be shown next to the cursor simultaneously, for each of selected routes. This design would allow researchers to analyze different routes' performances and compare them with other at the same time. It would also be beneficial to be able to compare different routes with each other within the same modeling, for instance, compare all routes under dynamic programming modeling. Or, to compare the same route under two different models of dynamic programming and cruise control.

It is noteworthy to mention that in these diagrams, the range of x-axis must be able to be filtered to only cover the desired period of time of the route (instead of the whole route). This is due to the fact that for some routes the number of data points is large.

In addition, page ”Graphs” must provide a feature that the user can save these graphs in different formats such as PDF and JPEG, or download the complete output in CSV or JSON format.

In the second set of figures of this page (as shown in the Result Section of this chapter, Figure 3.29), for each route, there must be a figure in which the velocity of the vehicle for both Cruise Control and Dynamic Programming is displayed in the same plot. Note that the x-axis in these plots represents the distance of the vehicle from the origin. Five other similar figures must also be shown; for gear, throttle, brake, acceleration and road slope.

Figure 3.28: A snapshot of the final version of the page ”Graphs” page. Once the web application is used for an origin-destination pair, a user can select, from the drop-down menu on the top-left corner of the page, up to six routes to simultaneously compare their performances over time in terms of velocity, gear, throttle, brake, acceleration, road grade, and distance through the course of the trip. When the cursor hovers over one point in time, the exact values of selected routes appear next to the cursor, simultaneously. A button on the top-right corner of this page allows users to download all these data for further analysis. For this figure, two routes are selected by a user.

For plotting the first set of figures (described above), we have written a JavaScript code in the front end that uses amCharts.js [25] library of JavaScript to plot the graphs.

Once this page is entered, a Python code (that we have written) runs in the back-end. This code once running uses the library Matplotlib [26] to generate the second set of figures (described above) and save them in JPG format. The front end then shows these plots on the page ”Graphs.”

Figures 3.28 shows a snapshot of the final version for the page ”Graphs” where users can interact with or download the data.

ω_1	ω_2	ω_3	ω_4	mass	min acc.	max acc.
1	1	1	1	25,000	-2.0	+2.0

Table 3.4: Values of Action Parameters and Objective Function Coefficients

Model	**MPG**	**Travel Time**	**Fuel**
CC	5.34	66 mins	11.93 gallon
DP	5.81	72 mins	10.95 gallon
Change	9%	+6 mins	-8%

Table 3.5: Fuel Consumption for a Route from Auburn, AL to Franklin, GA

3.4 Results and Discussions

3.4.1 Intrastate

Example 1. The web application is run for a 63.7-mile route from Auburn, AL to Franklin, GA and for vehicle parameters reported in [2] as the parameters of a typical Scania Heavy Truck. The coefficients of the objective function and the actions parameters are reported in Table 3.4

For this route, the fuel consumption rate, travel time, and total fuel consumption are reported in Table 3.5 for both the eco-driving (dynamic programming) and cruise control models:

In this example, the eco-driving returned by the web app will have a fuel consumption of 10.95 gallons (rate of 5.81 MPG) as opposed to the Cruise fuel consumption of 11.93 gallons (rate of 5.34 MPG). The fuel consumption saving is significant and is more than 8%. In exchange, the trip time of the eco-route is only 6 minutes more than that of Cruise Control trip time.

For completeness, we have also illustrated the details of a 10-mile portion of this route in the Figure 3.29. There are six sub-figures in the figure. The top left sub-figure illustrates the speed profile of the dynamic programming model in orange dotted-line and the speed profile of the cruise controller in blue solid-line. As the dynamic programming looks ahead it can adjust its speed in a way that it consumes less fuel. For instance, since the distance of 41 miles, the speed limit is less than distance 40 miles, dynamic programming slows its speed down much earlier than a cruise controller does, saving some fuel. The rest of the sub-figures illustrates the details of gear, throttle, brake, acceleration, and road's grade for both models. These sub-figures can be used for further research and analysis.

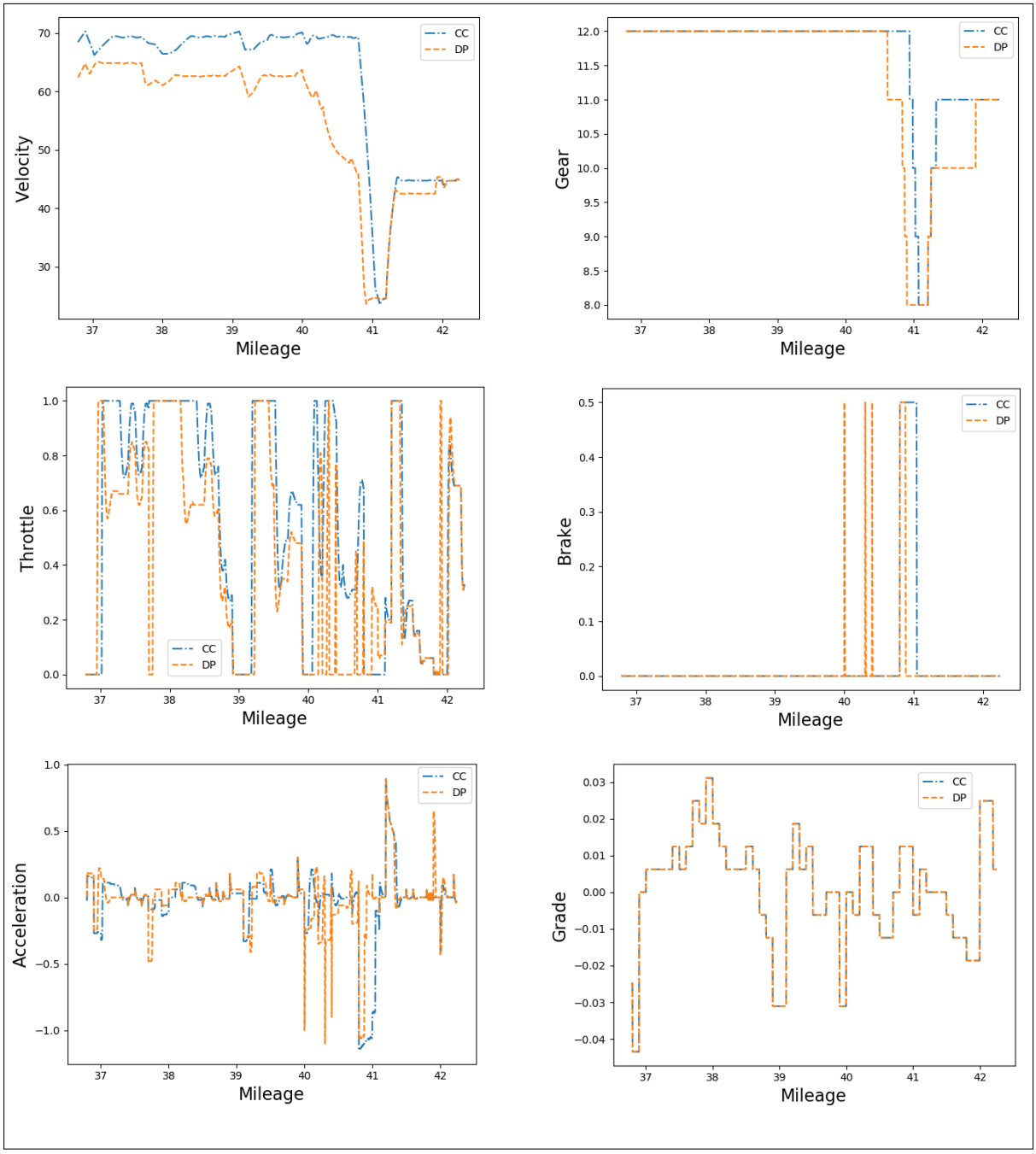

Figure 3.29: Auburn, AL to Franklin, GA. There are six sub-figures corresponding to velocity, gear, brake, acceleration and road's grade profiles for a 10-mile portion of the route.

Example 2. Figure 3.30 shows the "Summary" page of our web application when the origin is Auburn, AL and destination is Opelika, AL. This is another example for an intra-state route. As we can see in this figure, three routes are identified, and their information is summarized. The second route is the shortest in distance, time and fuel consumption, and has the highest miles per gallon. For the second route, the comparison between cruise controller

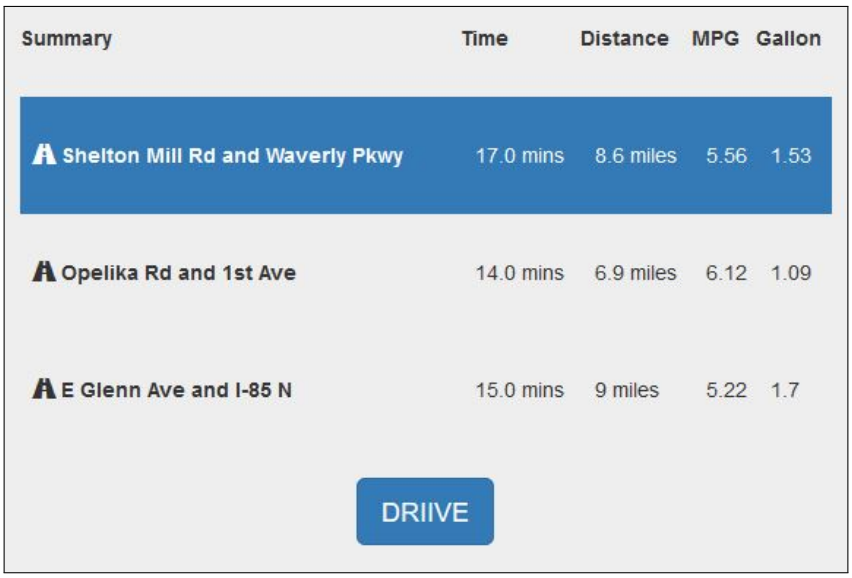

Figure 3.30: Summary Page for Auburn, AL to Opelika, AL. Three routes and their travel time, distance, MPG, fuel consumption are illustrated for the dynamic programming model.

and dynamic programming results show a 2% increase in miles per gallon while the total trip time has insignificant increase.

3.4.2 Interstate

Figure 3.31 shows the ”Summary” page of our web application when the origin is Auburn, AL and destination is Atlanta, GA. This is an example of a route between states. As we can see in this figure, three routes are identified, and the information is summarized. The first route is the shortest in distance, time and fuel consumption. We have shown, for the first route, the comparison between cruise controller and dynamic programming results in figure 3.32. As we can see, the miles per gallon has increased by 10% whereas the total trip time has increased by only thirteen minutes.

Also, from Figure 3.31, the second route is shorter in time than the third route. This is while the fuel consumption of the second route is more than that of the third route. The takeaway message from this example is that a faster path is not necessarily an eco-route.

To show that our web application is taking information on a real-time basis, we run our web application for the same origin-destination of Auburn to Atlanta, but at a different time than the time we got the results of 3.31. The result of this new run is shown in Figure 3.33. As

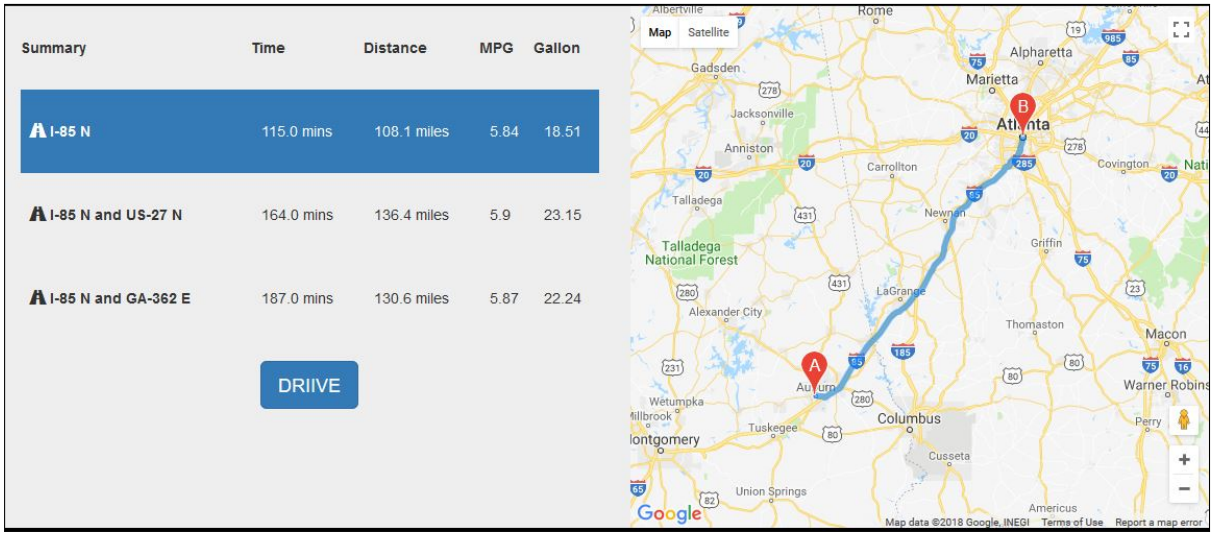

Figure 3.31: A Summary of Three Routes for Auburn to Atlanta

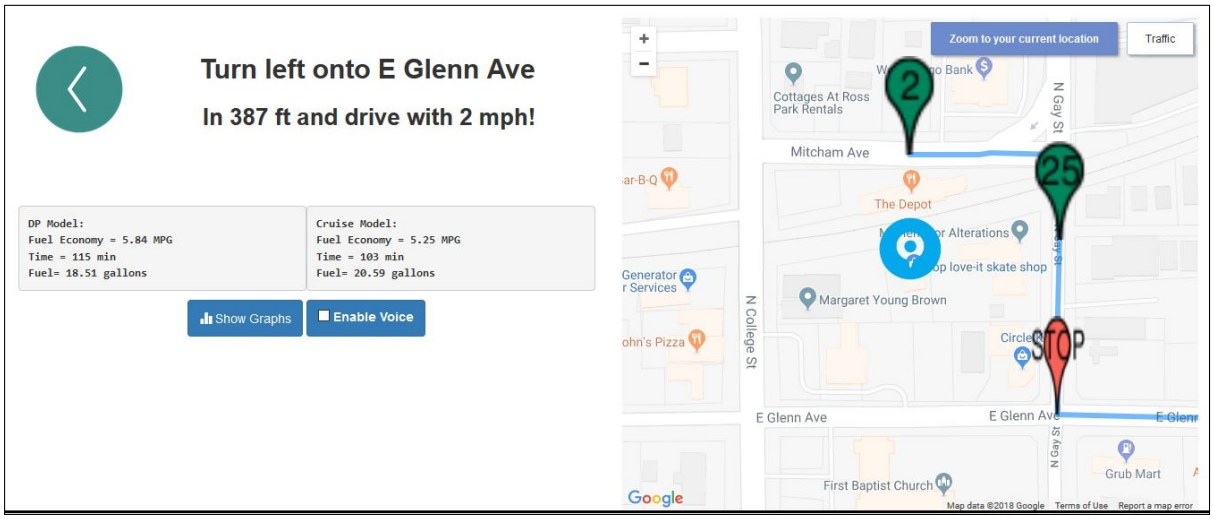

Figure 3.32: Comparison of cruise controller and Dynamic Programming Results for a Route from Auburn to Atlanta

one can see, this time, only two routes are displayed (as opposed to the three routes of Figure 3.31). This is because our web application works on real-time information available at the time the application is running. The ”Summary” page for this run is also shown in Figure 3.33 which is different from that of Figure 3.32.

3.4.3 Effect of Road Information and Weather Conditions

To show the effect of weather conditions in the model, we have run our web application for the same route of Auburn, AL to Atlanta, GA in two different times with two different weather conditions. One run is when the weather in all locations of the route is clear and the other time,

Figure 3.33: Auburn to Atlanta - Second Try. Two routes are returned, instead of three routes shown in Figure 3.31. This is because the web application takes real-time information.

Weather		Clear		Moderate	rain	
Model	**MPG**	**Time**	**Fuel**	**MPG**	**Time**	**Fuel**
CC	5.25	103	20.59	5.78	113	18.73
DP	5.84	115	18.51	6.12	121	17.67
Change	+10%	+12	-11%	+6%	+8	-6%

Table 3.6: Auburn to Atlanta with Different Weather Conditions

the weather condition during the whole trip is moderate rain. Our web application uses the speed limit ratio of 1.0 and 0.9 for clear and moderate weather, respectively (from Table 3.2). The results are shown in Table 3.6.

In the case of a cruise controller model, travel time in moderate rain weather condition is 10% longer than clear weather while fuel consumption is 9% smaller. The most portion of the route from Auburn, AL to Atlanta, GA is a highway with speed limit of 70 mi/hr. A moderate weather condition will decrease this speed limit to 70*0.9=63 mi/hr. Driving with 63 mi/hr is more efficient than 70 mi/hr.

Unlike a cruise controller case, the dynamic programming model does not necessarily drive with speed limits all the time. That is why the increase in travel time when the weather is moderate rain is only 5% longer than clear weather. With the same logic, the increase of fuel consumption was 5% which is lower than the increase in a cruise controller model.

The final observation from this table is that fuel consumption for dynamic programming with clear weather is larger than Dynamic programming with moderate rain. Even though any feasible solution in the ”moderate rain” is also included in the ”clear rain”, but since the total time is also a factor in the objective function, the dynamic programming model ended up with having higher fuel consumption with shorter trip time as the best solution.

Row	Model	Ave. Speed	MPG	Time	Fuel	Fuel Save
1	DP	31.2 mi/hr	6.81	74	5.66	–
2	CC	32.8 mi/hr	5.79	71	6.64	-15%
3	CC	32.6 mi/hr	5.88	71	6.55	-13.5%
4	CC	32.4 mi/hr	5.99	72	6.43	-12%
5	CC	31.7 mi/hr	6.11	73	6.30	-10%
6	CC	31.2 mi/hr	6.29	74	6.12	-7.5%

Table 3.7: Effect of Road Information on Fuel Saving

The resultant fuel saving, and more generally the saving in the objective function, of the dynamic programming can be attributed to three main factors: (1) knowledge of road elevations; (2) knowledge of stop signs locations and speed limits; and (3) driving at a lower speed (on average). To better understand the contributions of each of these three factors, we have run our web application for a route from Washington, DC to Baltimore, MD for multiple cases. The first set of cases addresses the contribution of the third factor, i.e., driving at a lower speed. To this aim, we run our application for the cruise controller with different average speeds. The results are shown in Table 3.7, the last column of which is the relative difference of fuel consumption of cruise controller model with respect to that of the dynamic programming model. From row two to row six of this table, the average speed of cruise controller is reduced such that its total travel time approaches that of dynamic programming. As one can see, as the average speed lowers, the fuel consumption of cruise controller increases. As shown in row six of this table, when the total travel time of cruise controller is the same as that of dynamic programming, the fuel consumption saving of dynamic programming relative to the cruise controller is 7.5%. This implies that 7.5% $(15\% - 7.5\%)$ of the fuel saving of dynamic programming compared to the cruise controller can be attributed to the factor three, i.e., driving at a lower speed on average.

But, even when the total travel times (and consequently average speeds) of the dynamic programming and cruise controller are the same, there is a gain of 7.5% in fuel saving if one uses dynamic programming. To understand how much of this saving is owed to the first factor, i.e., knowledge of road elevations, we have run our web application without providing it with road elevations information. The results show that in the case of no knowledge of road elevations, the saving in fuel consumption of dynamic programming relative to cruise controller

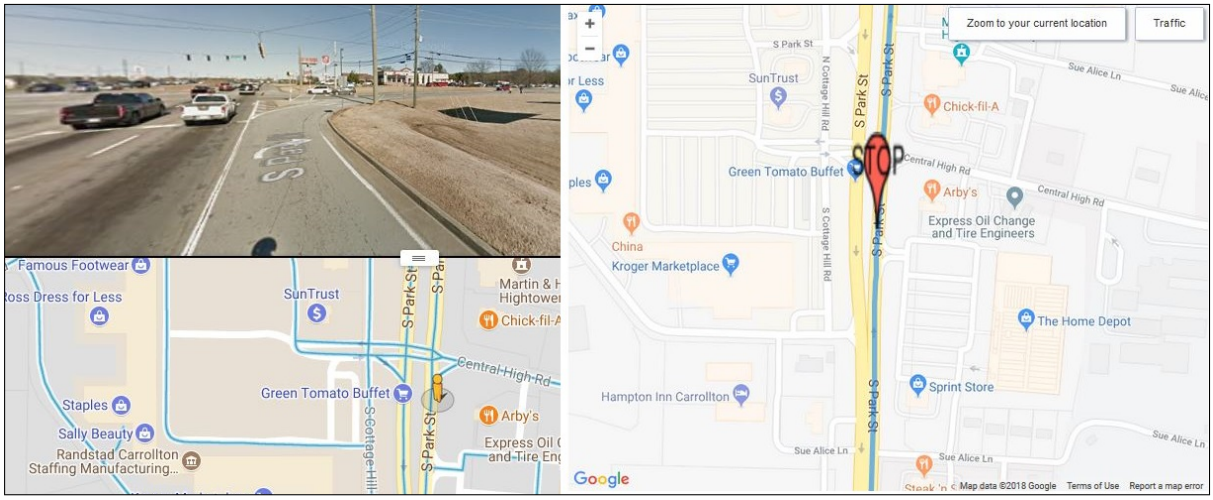

Figure 3.34: The Web Application Identifies a Traffic Signals

is 2% which is attributed to the factor two, i.e., knowledge of stop signs locations and speed limits. This, on the other hand, implies that 5.5% of the fuel consumption is attributed to the knowledge of road elevations.

3.4.4 Testing Road Selection Algorithm

Figure 3.34 shows part of Auburn, AL to LaGrange, GA route. The left sub-figure is a snapshot of Google Maps which shows a traffic sign ahead. As shown in right sub-figure of this figure, the web application correctly identifies the traffic signal as shown on the map with a ”STOP” icon.

Figure 3.35 shows part of Auburn, AL to Opelika, AL route where the vehicle should exit. As shown in this figure, the web application correctly identifies the first stop sign, then it increases the speed to 27 mi/hr. It then correctly identifies the exit route and on the exit ramp slows the speed down to 18 mi/hr, and then speed up to 43 mi/hr to merge to GA-27.

Figure 3.36 shows part of a route which has to go through a complicated area with a lot of highways. In this case, the algorithm correctly identifies the first stop sign (shown in the map with a ”STOP” flag, it then increases its speed to 31 mi/hr, and then again slows the vehicle’s speed down to 25 mi/hr. At this point in the route, there is a stop sign, right after which there is another stop sign. Our web application cannot identify the first stop sign. This is either due to

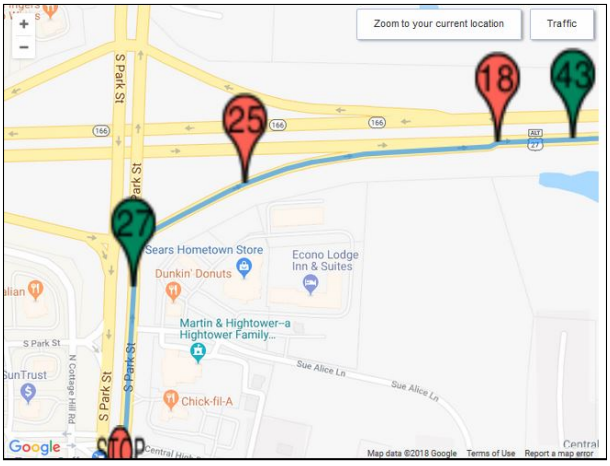

Figure 3.35: The Web Application Identifies a Highway Exit

Figure 3.36: The Web Application Misses One of the Stop Signs

the lack of information in the OSM database about the first stop sign (the one that is missed) or due to the imperfectness of our algorithm that cannot identify the first stop sign.

3.4.5 Sensitivity to Dynamic Programming Parameters

As described in section 2.2.3, the objective function of the dynamic programming has four coefficients that users can specify to reflect their preference between time, fuel consumption, velocity change, and brake (equation 2.16). It is expected that some change in these parameters will have some influence on the final results of the dynamic programming and consequently on the selected route. The default values of these four parameters are 1(coefficient of time), 1.0(coefficient of fuel), 1.0(coefficient of change in speed), and 1.0((coefficient of break).

We have selected a route from Auburn, AL to Alexander City, GA with distance 39.3 miles through the US-280 W highway. For this route, we conducted some analysis on the sensitivity to the four coefficients (of equation 2.16) of the trip time in minutes, trip's overall

fuel consumption in gallons, and the average miles per gallon of the whole trip. The results are shown in Table 3.8.

The first row of the table is related to the cruise controller case with no dynamic programming. The results of dynamic programming with default parameters are shown in the second row of the table. It is clear that the travel time has increased in exchange for better fuel consumption by applying dynamic programming.

In the third row of the table, the importance of Time is increased by increasing its coefficient to 2.5 (from 1.0) while keeping other parameters intact. The results confirm the expectation in that the travel time has decreased from 45 minutes to 43 minutes, but instead, the fuel consumption has increased.

In rows four and five of the table, the importance of fuel consumption is increased by increasing its coefficient to 2 and 5 (from 1) while keeping other parameters intact. The results confirm the expectation in that the travel time increases from 45 minutes to 47 and 54 minutes, but instead, the fuel consumption has improved.

In row six of the table, the importance of velocity change is increased by increasing its coefficient to 2.5 (from 1.0) while keeping other parameters intact. It appears that there is no significant difference in outputs when this coefficient is increased from 1.0 to 2.5. We have to further increase this coefficient to 5.0 in row eight of the table to be able to observe a noticeable change in the outputs.

We also decreased the importance of velocity change in row seven of the table by decreasing its coefficient to 0 (from 1.0) while keeping other parameters intact. Applying this change results in fuel consumption increase as expected. This is attributed to the fact that in the case that coefficient of velocity change is zero, there is no constraint on the speed change which means more acceleration (more throttling and more breaks) which results in increased fuel consumption.

Increasing the penalty of Velocity change resulted in a more fuel-efficient model with MPG of 6.30 (row 8). The high penalty of velocity change forces model to minimize the acceleration.

Row	Model	Fuel	Time	V.	Brake	MPG	TT	Fuel	T. Dif.	F. Dif.
1	CC	-	-	-	-	5.70	41	6.87	0	0%
2	DP	1	1	1	1	6.13	45	6.40	4	-7%
3	DP	1	2.5	1	1	5.82	43	6.73	2	-2%
4	DP	2	1	1	1	6.41	47	6.12	6	-11%
5	DP	5	1	1	1	6.72	54	5.84	13	-15%
6	DP	1	1	2.5	1	6.18	45	6.35	4	-8%
7	DP	1	1	0	1	5.93	45	6.61	4	-4%
8	DP	1	1	5	1	6.30	48	6.22	7	-9%
9	DP	1	1	1	5	6.19	45	6.33	4	-8%
10	DP	1	1	1	0	6.03	44	6.50	3	-5%
11	DP	1	1	0	0	5.90	45	6.64	4	-3%

Table 3.8: Sensitivity Analysis to the Dynamic Programming Parameters. CC: cruise controller, DP: Dynamic Programming. Columns Fuel, Time, V Change, and Brake are the coefficients in the objective function of dynamic programming. For a cruise controller model, there is no objective function. Hence no coefficients are specified. TT: Travel Time, MPG and Gallon are the outputs of the model corresponding to each set of parameters.

In rows nine and ten of the table, we analyze the dynamic programming model on brake level parameter. In row nine, the penalty for braking is very large, while in row 10 no penalty is considered for braking. The results show 1% increase in fuel consumption and 1.5% decrease in travel time as the penalty gets larger. Having smaller penalty on the brake may let the model have hard deceleration which is not very fuel-efficient.

Finally, in the last row, the parameters related to velocity and gear are set to zero to only analyze the importance of the main factors of Fuel and Time. The results show a less fuel-efficient model than the model with all four factors but still more fuel-efficient than cruise controller.

The overall conclusion from the above results confirm the expectation that increasing the coefficients of fuel, velocity change, and brake would lead to an improvement in fuel consumption, but also an increase in the travel time, whereas the effect of time coefficient is reverse, i.e., an increase in the coefficient of time would lead to a better travel time in exchange for a worse fuel consumption.

		DP			CC		
Row	**Mass(kg)**	**MPG**	**TT(mins)**	**Gallon**	**MPG**	**TT(mins)**	**Gallon**
1	15,000	7.31	42:25	5.36	6.93	40:57	5.66
2	25,000	6.13	44:56	6.4	5.70	41:18	6.87
3	30,000	5.63	45:35	6.96	5.23	41:40	7.5
4	35,000	5.32	47:01	7.37	4.84	42:14	8.1
5	40,000	5.01	48:19	7.83	4.52	42:50	8.67

Table 3.9: Sensitivity Analysis of Route Selection to Action Parameters. CC: cruise controller, DP: Dynamic Programming. Column Mass is the mass of the vehicle. TT: Travel Time, MPG and Gallon are the outputs of the model for a vehicle with specified weight.

3.4.6 Sensitivity to Action Parameters

In the fuel consumption model (2.2.2), vehicle's mass weight is an important factor determining the amount of fuel consumption. In table 3.9, we have run our web application for different vehicle weights for the same route from Auburn to Alexander City. For both the cruise controller (CC) and the dynamic programming (DP) cases, as the weight is increased, the travel time and fuel consumption increase, as expected.

3.4.7 Comparison with Existing Methods in the Literature

As mentioned in the introduction, the Dynamic Programming model is one of several approaches that have been used to solve the velocity trajectory problem. Other methods include, but are not limited to, Sequential Quadratic Programming models, Adaptive Nonlinear Model Predictive Controller algorithms, Shooting algorithms, and Analytical Solution approaches.

As has been shown in the previous sections, our approach can result in a good range of fuel consumption savings of up to 15% if one allows for an insignificant increase in the travel time, and up to 7.5% of fuel consumption savings if no increase in travel time is allowed. The fuel consumption savings depend highly on the input parameters, road elevations, speed limits, weather conditions, and user preferences.

Our results are consistent with the results of other studies conducted in this area and exceed some of those results in certain cases. One advantage of our work is that our results are all based on real-time information about roads and weather, and unlike some other papers, do not depend on assumptions, unrealistic parameters or information.

In the following, we briefly review the results of some of the papers published in this field for comparison purposed.

In [27], the authors develop an analytical solution to the optimal velocity trajectory problem while incorporating road grades into consideration. The authors compare their solution with a dynamic programming model and a cruise controller and show that their solution and dynamic programming model is 8 to 10 percent more efficient than a cruise controller on a road with fictitious elevation profile. They further claim the speed of their analytical solution is much faster than the dynamic programming speed.

In [28], the authors develop an Adaptive Nonlinear Model Predictive Controller (ANMPC) algorithm to solve the optimal velocity trajectory problem. The authors assume road information such as elevations are accurately known and take advantage of Sequential Quadratic Programming literature to approximate a solution to the ANMPC problem. Their results show a 2.4% fuel saving compared to a cruise controller.

In [29], the optimal speed trajectory problem is studied assuming in-advance knowledge of road grades. The authors formulate the problem as an open-loop optimal control problem and solve it numerically using a multiple shooting algorithm. The Pontryagin minimum principle and Karush-Kuhn-Tucker conditions are used in their paper. The authors show, through simulation, that a fuel savings of 4.6% is achievable using their approach.

In [17], an algorithm is developed based on a dynamic programming model to solve the fuel-optimal control problem. Two main focuses of this work are gear shifting modeling and the low computational complexity. The authors of this work show a 3.5% decrease in fuel consumption in their method relative to a cruise controller model.

In [30], the authors propose a control model for reducing drivers' fuel consumption. The proposed method is tested using microscopic traffic simulator AIMSUN. The results of their tests show a 13.2% fuel consumption saving in exchange for a 16.44% time increase.

3.4.8 Discussion

It is worth noting that the results reported in this study are initial results and are limited by a few factors. (1) The output of this application which is the optimal speed profile may not be

necessarily and fully followed by drivers. This could be due to the traffic intensity of the road or due to unpredicted events that may take place on the road. Furthermore, other drivers' behavior may also prevent the driver to follow the speed profile exactly. For example, a driver cannot drive as the web application instructs if its lead vehicle is driving slowly. Therefore, the amount of fuel saving may decrease when the speed profile is not exactly applied. (2) The parameters which are used in the fuel consumption model may not perfectly represent the engine's fuel consumption of a vehicle, and therefore, the reported fuel savings could be slightly different. (3) The OpenStreetMap database does not contain the speed limits of all the roads in the United States. In the case that OpenStreetMap does not have a speed limit for a specific road, a default speed limit that is based on the type of that road is selected, which may not always be accurate. Therefore, fuel consumption may change when a speed limit is missing in the OpenStreetMap database.

These limitations are not specific to this study and are applicable in general to the results of papers existing in the literature on this topic.

3.5 Conclusion and Future Recommendations

In this chapter, we have improved our web application significantly. Our web application can be used for city or highway driving. It takes routes speed limits, traffic signs, and road elevations into consideration when finding the optimal speed profile. It further incorporates real-time weather information to adjust the speed limits such that drivers safely are considered. Considered are also the vehicle characteristics that users can provide the application with such that the speed profile guidelines are specific to their vehicles. For convenience, this information needs to be entered into the application only once and will be stored for future use. We have also improved and enhanced our web application further to make it more practical and safer to use by enabling voice and visual guides.

There are some areas in which our web application can be more developed. The first area is the speed of our web application. The speed is mainly capped by the dynamic programming algorithm's speed. As with any dynamic programming model, there is always a trade-off between the dynamic programming speed and its optimal solution. The more the speed and

distance variables of the dynamic programming are discretized, the slower its speed will be, in exchange for a more optimal solution. Since our web application is a real-time app, any improvement in efficiency of the algorithm will be appreciated.

Another area of improvement is the road selection algorithm. As we have shown, there are a few cases in which our algorithm misses a traffic sign, or select a wrong speed limit. More sophisticated algorithms should be considered to prevent such mistakes, but care should be taken about the time these more sophisticated algorithms take to run.

Chapter 4

An R-based Web Application for Mining GPS locations data

4.1 Introduction

Thanks to the advanced global positioning system (GPS) technology, keeping records of locations of moving vehicles is nowadays a common practice and is done either through wearable devices such as smartphones or by installing GPS devices on vehicles.

GPS locations databases, in their raw formats, are difficult to be used for optimization and statistical analysis because of the following issues that are inherent to them:

- The size of these databases is very large, and in fact too large for any statistical analysis to be performed on them in a timely manner.
- Due to the high frequency of location sampling, these databases contain a lot of conditionally redundant data, whose presence renders conducting any statistical analysis on them very cumbersome, and whose absence causes information loss of little (if any) significance.
- In these databases, information of locations are provided, and trips are not readily identified. This is undesirable for further data analysis because a user cannot determine which truck took which trip at what time.

In this chapter we describe and develop an R-based web application which automate the following:

1. Construction of a new trip-based database that addresses the aforementioned issues of the original database. This new database is trip-based, of a much smaller size well suited

for fast data mining and statistical analysis, in which conditionally useless data are significantly removed. Key fields of the new trip-based database are date and time of the trip, trip origin, trip destination, the elapsed time of the first stop immediately after this trip, truckID or driverID of the trip.

2. Enriching the constructed trip-based database with the weather conditions that a specific trip encountered.

3. Enriching the newly created trip-based database with the information of Accidents and Incidents that took place during a specific trip.

4. Generation of several interactive visualizations that describe different characteristics of the trip-based database.

The aim of this web application is not to do statistical and optimization analysis, but to provide the user with a database that is containing very useful information for statistical and optimization analysis. In addition, this web application provides some useful interactive visualizations corresponding to the newly created database.

The GPS locations databases do not yield semantic information about trips. Hence, the first and fundamental step is to identify trips from the GPS locations database. Trip identification and origin and destination stop locations identifications are two equivalent problems, and therefore, we will use them interchangeably through this chapter.

The general question of identifying trips or, interchangeably, stop locations has appeared in literature in a variety of research works, and has been posed in different forms. In [31], the identification of activity stop locations in GPS trajectories is studied as the main underlying research problem. The authors categorize the stop locations into two main groups of non-activity stops and activity stops, and propose a combination of density-based clustering method and support vector machines to determine the location of activity stops. In [32], this question is addressed for crowd location forecasting. The authors propose a spatio-temporal prediction approach. In [33], the question of stay(stop) location identification has come up in the broad research field of Active and Assisted Living (AAL). The authors of this work introduce a framework for the evaluation of stay detection methods. The authors then use this framework to test

and evaluate different stay location identification methods. In [34], a related question of detecting driving cycle from wearable GPS data has been studied. In [35], the problem of extracting stops and trips from individual people's GPS locations is addressed by analyzing two main factors of non-movement periods and longest movement periods.

In terms of trip and stop location identification problem formulation and the approach used to attack this problem, the works of [36] and [37] are closest to, yet different from ours. In [36], the problem of converting GPS data to travel data is studied, in which the authors develop an algorithm to detect non-movement periods first, and then compare the duration of such non-movement period to a dwell-time threshold; if the duration is more than this threshold, the non-movement period is determined to be a stop for the vehicle. To complement their method, the authors use the implementation logic of [38], and in addition, check the vehicle's speed during this non-movement period ([39]). The method used in [36] falls in the more general approach of threshold-based detection. Another example of the research work done in the area of threshold-based detection is the work of [40]. The aforementioned dwell-time in threshold-based detection is a tuning parameter that varies from 120 seconds ([41]) to 600 seconds ([42]).

The work of [37] is the main source that introduced a formal model for incorporating points of interest (PoI) in the problem of trip and stop identification. The PoI had been used in the trip and stop identification problem earlier, but had not been formulated in a systematic and methodical manner prior to the work of [37]. The PoI are the places that are of interest to the user, and therefore is application based. Examples of PoI's are hotels, malls, restaurants, or even traffic signals. Multiple databases of PoI's are first constructed, and then based on the application, some of them are used to help increase the accuracy and speed of trip and stop identification problem. Many researchers have extended the work of [37], an example of which is the work of [43] in which the authors propose some ranking for the PoI's based on historical data.

Our method in identifying the trips from GPS locations database differs from that of [36] and [37]. We divide the whole map into a significantly large number of small non-overlapping areas and measure the amount of time it took the vehicle to leave one area once it enters it. Based on this elapsed time we decide whether there has been a stop in this area. Another

difference is the underlying problem. In our work, the underlying problem is to find and identify a trip with its major stops. The work of [36] and [37], however, have different aims. For example, the work of [36] aims to find all the stops, even the ones with very short elapsed time in the order of minutes.

The rest of this chapter is organized as follows. In Section 4.2, all modelings and methodologies that are used in this chapter are described. In Section 4.3, our R-based web application are described and developed. In Section 4.4, the results and some discussion are provided. Finally, Section 4.5 concludes this chapter.

4.2 Modeling and Methodology

In this section, all databases that are used in this chapter are described. In addition, the trip-based database that our R-based web application generates along with the algorithm used in this process is detailed. Through the rest of this chapter, without loss of generality, we use "truck" to represent a moving object with a GPS system.

4.2.1 Locations Databases

GPS locations database of the truck, Accidents locations database , and Incident locations database are described below.

GPS Locations Database

Records of GPS locations along with other related useful information are kept in a database to which we refer by GPS locations database. This is the main database which contains the coordinates of the GPS locations, the timestamp of recording, the truckID, the driverID, truck's speed, and movement direction of the truck.

As already pointed out, this database in this raw format is not suited for data analysis and needs to be converted to a trip-based database, i.e., a database in which all trips are readily identified.

Incidents and Accidents Databases

Sometimes, sensors are installed on a truck to determine if an incident has happened, in which case a record of the timestamp of this incident, the latitude and longitude of its location, and its type are stored in a separate database (Incidents database). One example of an incident can be a ”hard break” which happens when a truck decelerates very fast. The information contained in such database may be used for understanding each driver's driving behavior.

Another similar database (the Accident database) keeps records of all accidents in which a truck is involved. Such information contains details of the accident such as its timestamp, its location, type of accident, and some description of the accidents. These two databases of incidents and accidents can be used to determine if there has been an incident or accident in a specific trip. Constructing and maintaining these databases are common practices in some industries, such as the transportation industry.

4.2.2 Online Databases

Some useful information is missing from the original raw database, examples of which are weather-related information at the time and location of the trip. Weather information is important to researchers who want to understand and study the environmental factors a specific trip faces and their effects on the trip from different perspectives.

Another missing information is the accidents that have taken place during the time period of the trip and in a location in the vicinity of the trip, for which the truck of our interest was not involved. Having accident information that is specific to a route/trip will help researchers allocate a historical danger/risk measure for that specific route/trip.

To enhance the trip-based database generated by our R-based web application, we exploit related databases that are available online and add their information to our generated database as described below.

Weather

Weather information is not always included in the GPS location database. Therefore, we need to collect such information from another database. There is a capacity on the requests that can be made per minute to Weather API's. Therefore, we decided to create our own database from the data provided by National Oceanic and Atmospheric Administration (NOAA) ([44]).

NOAA provides hourly weather data for over several thousand stations around the world, for over the past 30 years. For each station and year, there is one specific file that contains the hourly weather information corresponding to that station in that year. For each trip, our web application will find the station closest to that trip and extract the weather information for that time of the trip from the weather file corresponding to that station.

An example (taken from NOAA) of the information for one hour of one station is given below.

```
0159008411999992015010114524+08200+134600FM-15+000099999V020060
1N005719999999N005000199+02841+02471100921ADDAW1051MA1100911999
999OC100771REMMET118MOBOB2METAR XM20 //000 082346 011452Z AUTO
06011G15KT 5000 HZ 28/25 A2980 RMK A02 SLP092 T02840247 PA112
DA2094=
```

NOAA provides a document in which it is explained how to read these records ([45]). More specifically, it explains what each character in the above example determines. For instance, characters 16 to 27 identify date and time (in this example, characters 16 to 27 are "201501011452" which is related to date 2015/01/01, and time 14:52).

We have written some codes in Python that extracts the information of interest from the above sequence of characters and stores it in a MySQL table for further use.

US Fatal Accidents

We have downloaded the Fatality Analysis Reporting System (FARS) database ([46]) for the past five years. This database keeps records of a lot of factors related to fatal accidents from

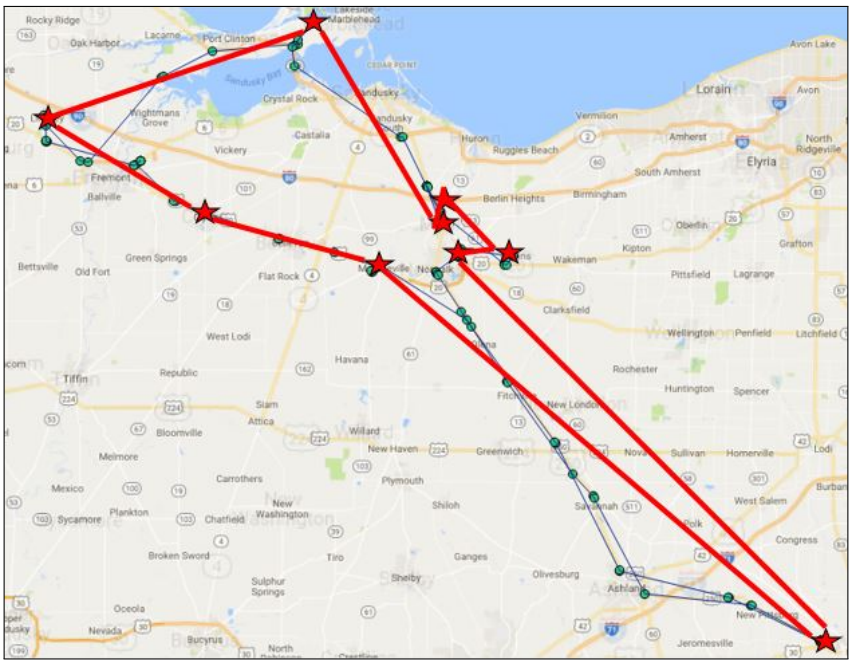

Figure 4.1: All locations of one truck for a given day are extracted from the GPS locations database and shown on the map using small green circles. Also, trips made by that truck for that day along with their destinations are illustrated by thick solid red lines and red stars, respectively.

1975. The time, location, and the number of fatalities of all accidents that led to death are of our interest in this research.

4.2.3 Trip-Based Database

As previously mentioned, the goal is to generate a trip-based database, in which each row corresponds to a specific trip and contains all information related to that trip.

In Figure 4.1 all locations of a given truck for a given day are extracted from the GPS locations database and shown on the map using small green circles. Based on this illustration, one can neither identify nor visualize the trips that the truck made during that day, let alone extracting more details about the trip.

The developed R-based web application first finds all destinations of the truck using an algorithm which is explained in the next section. Then, based on the destinations, all the trips this truck made on that day will be extracted. In Figure 4.1, destinations and trips are illustrated by red stars and thick solid red lines, respectively. It is clear based on this output that there are 9

trips with identified origin/destination locations of each trip. After each trip is identified, some more information about each trip will be collected and stored.

Each row of the created trip-based database has the following fields:

- Trip variables
  - tripID which uniquely identifies a trip
  - truckID which uniquely identifies a truck
  - date/time of the start of the trip
  - date/time of the end of the trip
  - latitude of the origin
  - longitude of the origin
  - latitude of the destination
  - longitude of the destination
  - driverID which uniquely identifies a driver
  - trip time which is the duration of the trip
  - stop time which is the duration of the stop before the next trip begins

  Note that from the aforementioned trip variables, only truckID and driverID are known in the original raw database. The other variables require data-mining: a trip must first be identified.

- Weather variables
  - temperature at the origin
  - wind speed at the origin
  - wind direction at the origin
  - visual distance at the origin
  - precipitation at the origin

  - weather condition at the origin

  Once a trip is identified, we data-mine from the NOAA database the weather information at the time and near the location of the identified trip.

- incident variables
  - number of incidents that took place during the trip
  - latitude of the location of incident
  - longitude of the location of incident
  - date/time of the incident
  - type of the incident
- Accidents variables
  - number of accidents that took place during the trip
  - latitude of the location of accident
  - longitude of the location of accident
  - date/time of the accident
  - type of the accident

The above variables are data-mined for each identified trip from the Accidents and Incidents databases if they are available.

4.2.4 Algorithm

As explained before, first and the most fundamental task is to identify a trip from a GPS database. Trips can be defined by a movement between two consequent destinations. The first thing that comes to mind in identifying the destinations is to find the location at which vehicle's speed is zero. There are some cases in which this simple algorithm does not work correctly. For example, vehicles may report the speed of zero when they reach a traffic signal or a stop sign, which are obviously not their final destination. Furthermore, the speed is reported by a

sensor which may not be accurate enough. After trying different algorithms, we decided on a heuristic algorithm. This algorithm works as follows. It first divides the whole area, that the GPS locations database covers, into a significantly large number of smaller non-overlapping areas. If the trip's destination is not in one small area, we expect that the moving truck stays in such small area no more than a few minutes, even in urban areas. If the reports from a truck show a longer stay, we suspect that the truck had a long stop somewhere inside the small area. In that case, we consider this small area as the final destination. A step-by-step description of the first phase of the algorithm is as follows:

1. Divide the area covered by the GPS locations database into a significantly large number of smaller non-overlapping areas. All locations whose latitude rounded down to two decimal points are the same and whose longitude rounded down to two decimal points are the same belong to the same area. Each of these areas is roughly one-mile-squared. For example, a location with latitude and longitude of (32.595783 , -85.478921) and a location with latitude and longitude of (32.596783 , -85.474921) both belong to the same area, because both locations have the same rounded latitude and longitude of (32.59 , -85.48).

2. Sort GPS database by truckID.

3. Sort the sorted GPS database by timestamp.

4. For each truckID and each specific area, find the timestamp the truck entered that area and the timestamp it exited that area.

5. If the time difference between the two timestamps is greater than a threshold, then that truckID has had a stop in that area with a high probability.

6. Repeat the above algorithm for all truckIDs

The next step in our algorithm is to collect weather data for the trips. The algorithm connects to the Weather database that we have created and reads the weather conditions corresponding to the trip location and time, and adds this information to its corresponding trip in the final trip-based database.

Step three in our algorithm is merging the incident database and the created trip-based database. The incident date and time is used to identify a trip that the corresponding truckID was in. The incident is then added to the identified trip. It may happen that more than one incident takes place during one trip. In this case, all incidents are shown, but separated with semicolons, in the incident field corresponding to that trip.

In step 4 of our algorithm, we connect to the accident database and take similar actions similar to those of step three in order to add the accident-related information to out trip-based database.

4.3 Web Application

As part of this project, a web application is developed that is based on the shiny package in R ([47]). In this section, a comprehensive description of our web application is provided. There are two panels on each page of our web application.

4.3.1 Home Page

In this page, some information about the project and its goals are given, the databases used in this R-based web application are described, instructions on how to use this web application and its features are shown in a short video, and a brief background of the developers of this web application is provided (Figure 4.2).

4.3.2 Upload Page

To start this web application, the user needs to upload the required databases first. There are two options, either to upload an already created trip-based database, or to upload a raw database (Figure 4.3). If the first option is selected, the user is asked to input the address of the file (Figure 4.4). Once the file is uploaded, the content of the file will be added to the trip-based database and the Summary Box is updated. Summary box summarizes the information of trip-based database which is located in the left side panel. More specifically, it shows the number of trips existing in the trip-based database as well as the number of incidents and accidents reported. It also shows how many trucks and drivers are in the database and finally the time

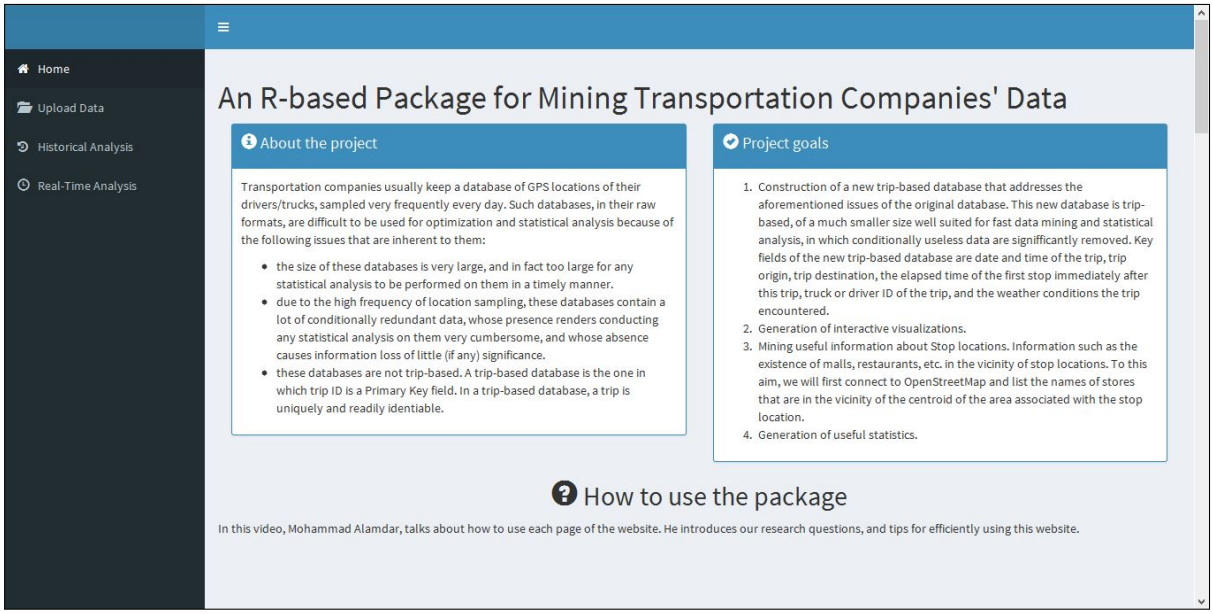

Figure 4.2: A screen-shot of the homepage of the developed R-based web application. This page provides a brief introduction to the project, its goals, and its developers. It also contains a short video on how to use this web application.

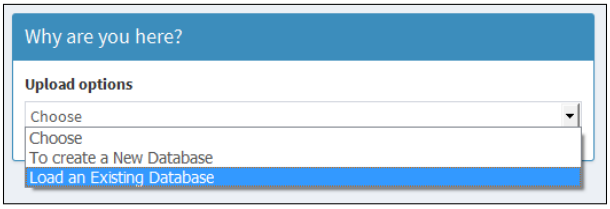

Figure 4.3: Database upload options - Step 1

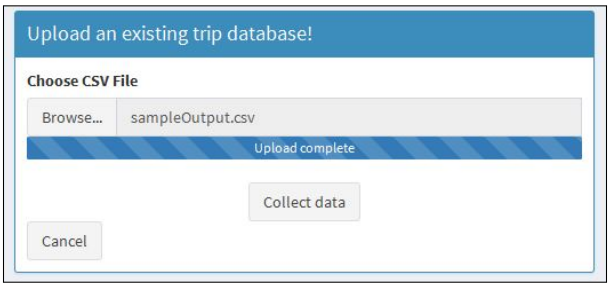

Figure 4.4: Users may upload an already created trip-base database.

period of this database (Figure 4.5). As one can see in Figure 4.5, the left side panel has the links to all pages of Home, Upload page, Historical Analysis, and Real-time above the summary box.

If the raw database option is chosen, two options appear: to read from a CSV file or to read from a MySQL file (Figure 4.6).

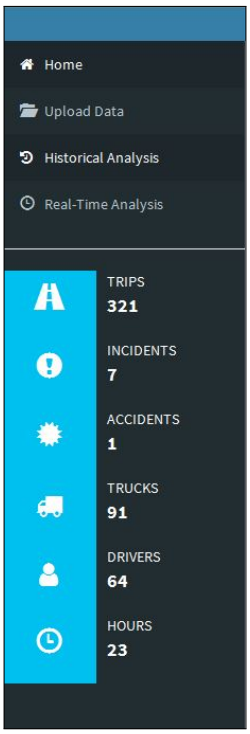

Figure 4.5: A snapshot of the slider panel located on the left side of the web application page. On its upper part, the slider panel has links to all pages of this web application, and on its lower part, a summary of the generated trip-base database is shown, which contains information on the number of trips, incidents, accidents, vehicles, drivers, and length of the time period covered in the database.

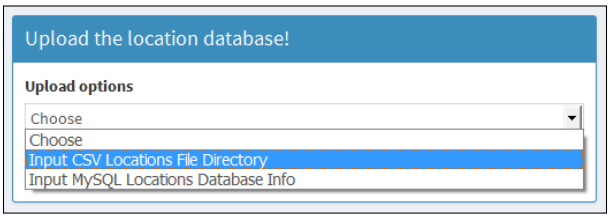

Figure 4.6: The raw database can be in a CSV file format or a MySQL format.

If CSV file is selected, a browse request appears. If MySQL is chosen, the user is asked to fill four fields: User, database, password, and table (Figure 4.7).

In either case of a CSV file or MySQL, a Collect Data button appears, which after being clicked will connect to the database. Once connected, the R-based web application reads all the data from the file. After the data is read, the user is requested to enter more information. The required columns are truckID, lat, lon, driver id, DateTime. For each option, a drop-down appears which shows the list of all columns of the uploaded file. The user then needs to match

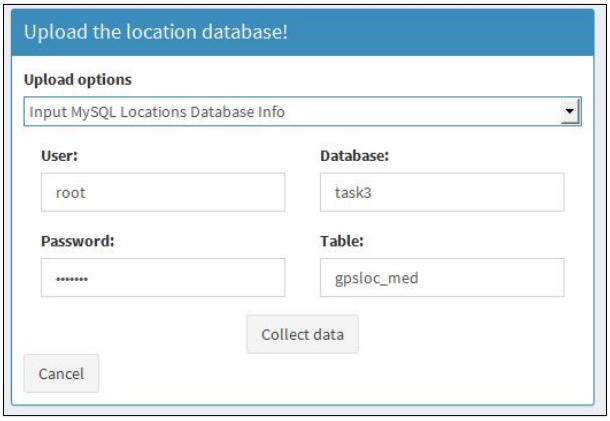

Figure 4.7: For MySQL format, users should fill four fields of User, database, password, and table.

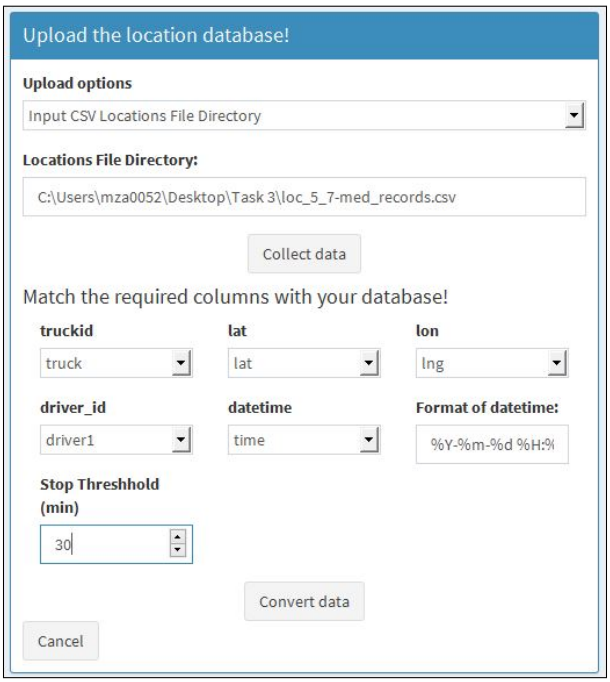

Figure 4.8: As part of the upload process, users are asked to match the required columns to their uploaded raw database.

a column from the drop down to the name of the field. For instance, for the field truckID, the user should select the name in the drop-down that corresponds to truckID (Figure 4.8).

The user needs to specify the format of DateTime field. For instance, the format of ”6/13/2018 21:8” is ”%m/%d/%Y %H:%M”. As another example, the format of ”2018-06-11 21:08:32” is ”%Y-%m-%d %H%M%S”. For more examples, see [48].

The last field is the Stop Threshold in minutes which is explained in Section 4.2.4. The user needs to set this field as well.

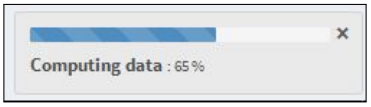

Figure 4.9: A progress bar showing the percentage of the constructed trip-base database.

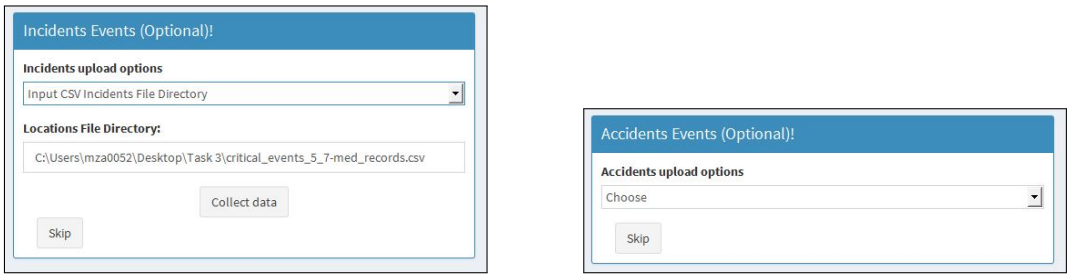

Figure 4.10: Uploading incident and accident raw databases

After all these fields are filled, the user needs to click on the ”convert data” button. An algorithm will then be applied to the uploaded database to create a trip-base database. The application also connects to the weather database using MySQL and collects the weather set of variables of the time and location of each trip. A progress bar appears on the lower left side of the page indicating the progress of constructing the trip-base database (Figure 4.9).

Once the new trip-base database is created, the GPS Upload panel disappears. and R-based web application goes to the next step which is to upload an incidents database. This step is optional, and therefore the user is allowed to skip this step. For the incidents panel, the user is asked to upload an incidents database, which our R-based web application will add to the created trip-base database. Similar to GSP Upload panel, users have two options of CSV or MySQL files for upload. ”Collect data” button needs to be then pressed, for the incidents data to be read (Figure 4.10). Similar to the previous process, in this case, we also need to match columns of the input data file to the required fields, which are truckID, lat, lon, type, DateTime. The format of DateTime needs to be specified by the user. These fields specify the date and time of the incident, its location’s lat and lon, type of the incident, and ID of the truck that was involved. ”Add” button needs to be clicked next to merge the incident database and the previously created trip-base database.

Once the incidents step is over, the third step starts, which allows users to have an option to upload an accidents database (Figure 4.10). The process here is the same as step 2. Once the

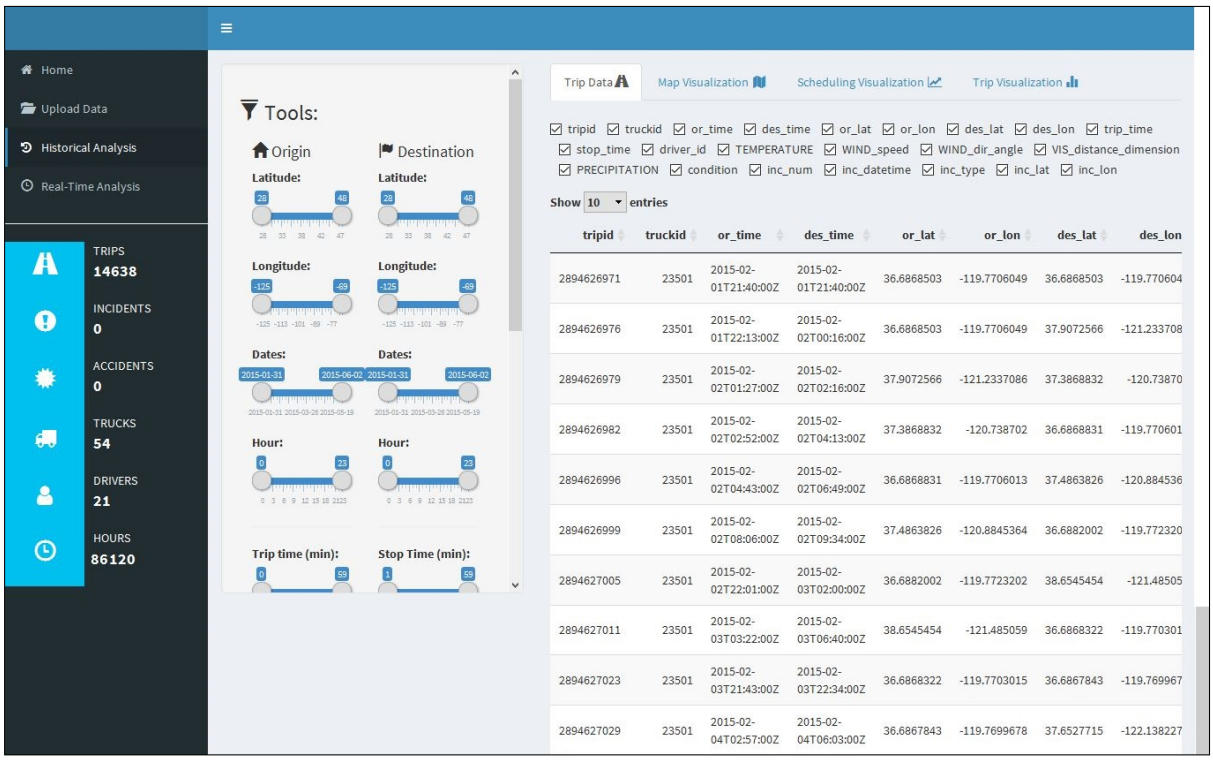

Figure 4.11: Trip data tab of the Historical Analysis Page. Data are illustrated in the format of a table. The table by default is composed of 22 columns. The user has the option to remove any column or sort the table based on any column. We have designed the left side panel such that it can be hidden if the user wants to see the table in full width.

third step is finished, the Summary Box is updated. Users can repeat all aforementioned three steps again to add a new set of databases to the current ones. This will help users to merge multiple databases.

4.3.3 Historical Analysis Page

In this page, the trip-base database that we have created is shown in a table format and under different visualizations that we describe in the rest of this section. There are four tabs in this page, which are Trip Data, Map Visualization, Scheduling Visualization and Trip Visualization. In trip data tab, data is illustrated in the format of a table. The table by default is composed of 22 columns. The user has the option to remove any column. The user can also sort the table based on any column they desire. This can be done by clicking on the column name (Figure 4.11). This page has a lot of contents. Therefore, we have designed the left side panel such that it can be hidden if the user wants to see this page in the full widths of the screen.

Map Visualization

In the map visualization tab, the information in the trip-base table is shown on an OpenStreetMap using the leaflet library for R ([1]). This visualization is interactive allowing a user to zoom in, zoom out, and interact in general with the map to visually navigate through a neighborhood, identify, and learn more about its popular locations. We believe that this visualization (Figure 4.12) is of significant use to different types of analytics such as Vehicle Routing Problems and Scheduling. The map has six different components which are listed as follows:

- **Green lines.** All trips are shown on the Map by a solid green line between the origin and destination of the trip. For each origin-destination pair, we have found the number of trips made between these two locations. This number is reflected in the thickness of the line connecting this origin to this destination, i.e., the thickness of the connecting line shows the intensity of the trips between this origin and destination in the time duration of the trip-base database.

- **Dark Shaded Squares.** On the map, each square in a dark shaded color is a one-mile-square area containing the locations of origin or destination of a trip. The color intensity and shade increases as the number of stops in that area increases; the more intense the dark color, the higher the number of trips that started or ended in that area is. The component provides a lot of useful information about where most of the stops occur and which stops are more popular.

- **Blue Dots.** There might be different stores and restaurants in each dark shaded square explained above. To have more information about which location in each square has higher stops, the centroid of all stop locations in each square is found and shown on the map by a black-filled small circle.

- **Yellow Circles.** The exact location of each reported incident is shown on the map by a yellow circle.

- **Brown Circles.** The exact location of each reported accident is shown on the map by a brown circle.

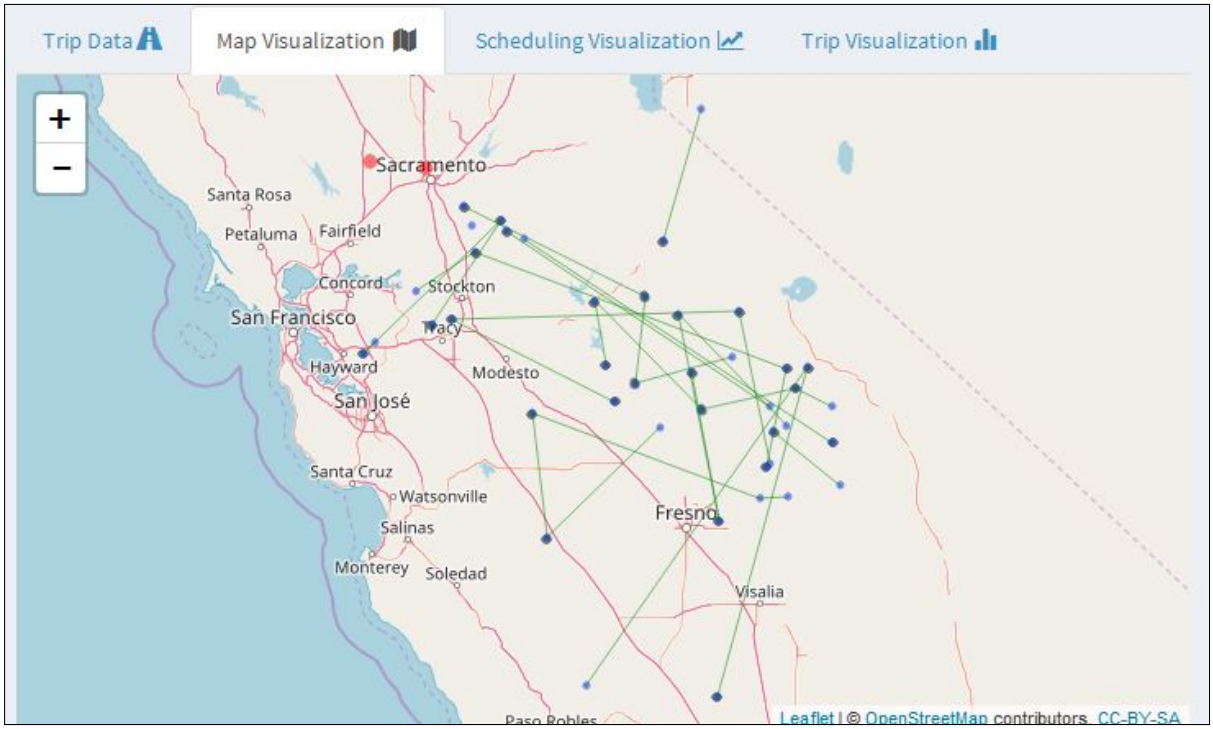

Figure 4.12: The information in the trip-base table is shown on an OpenStreetMap using the leaflet library for R ([1]). This visualization is interactive allowing a user to zoom in and out, and interact in general with the map. The map has six components of green lines, dark shaded squares, blue dots, yellow circles, brown circles, and red circles.

- **Red Circles.** We have used FARS to show on the map the location of the fatalities that occurred during the time period of our trip-base database.

Scheduling Visualization

The aim here is to visualize the intensity of moving trucks from different perspectives. There are five visualizations in this tab.

The first visualization (Figure 4.13) illustrates the hourly intensity of moving trucks for the given period of time. The x-axis is the hour in our time period, and the y-axis is the number of moving trucks in that hour. For instance, if our time period is 14 days, there are $336 = 14 \times 24$ points on this visualization. There is a lot of low-hanging, yet important and useful, information that can be extracted visually. For example, from this visualization, one can see which hours of the day are the most crowded in terms of the number of moving trucks and whether this hour is consistent through the days of the period. Or, whether there is a specific day in which the intensity of moving trucks in each hour is different from the rest of the days. This could

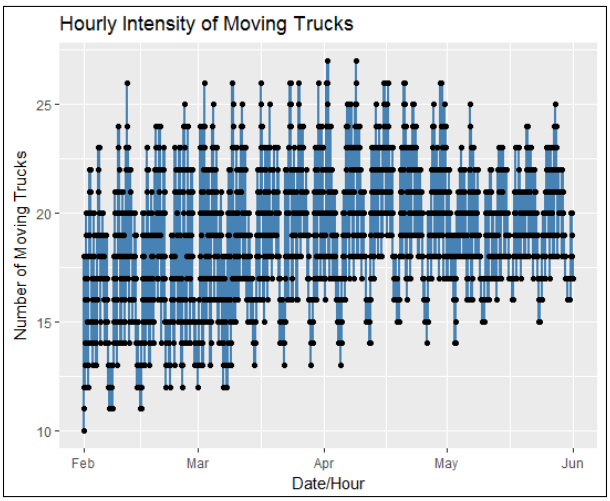

Figure 4.13: Hourly intensity of moving trucks. For each hour on the x-axis, the number of trucks moving during that hour is shown on the y-axis.

happen for example for holidays or weekends. This visualization is useful the most when our time period is short, i.e., at most a few weeks. If the time period is longer, users can shorten the time period by using the tools provided in the toolbox described later in this section.

A (x, y) point on the second visualization(Figure 4.14) represents the total number of moving truck-hours (y) during the x^{th} day of the year. For instance, if our time period covers 100 days, there will be 100 points in this visualization, each corresponds to one day in this time period. To calculate y-coordinate, we look into each hour of that day and count the number of moving trucks in that hour. We then count the total number of such moving trucks for 24 hours of that day. There is fruitful information hidden in such visualization that can be extracted. An example of such information is how the intensity changes over days of the given time period. This visualization is useful the most when our time period is no more than a few months. Similar to the first visualization, users can exploit the tools provided in the toolbox to filter the time period to their desired width that suits best to this visualization.

The x-axis of the third visualization(Figure 4.15) represents a specific week from our time period, and its y-axis is the total number of truck-hours over that week. The calculation of y-coordinate is similar to the one explained in the second visualization, with the difference being the accumulation period is an entire week, instead of a whole day. Such visualization would reveal if there are weekly trends in our data. This visualization comes into use in the cases when the time period is several months to a few years. Similar to previous visualizations, users

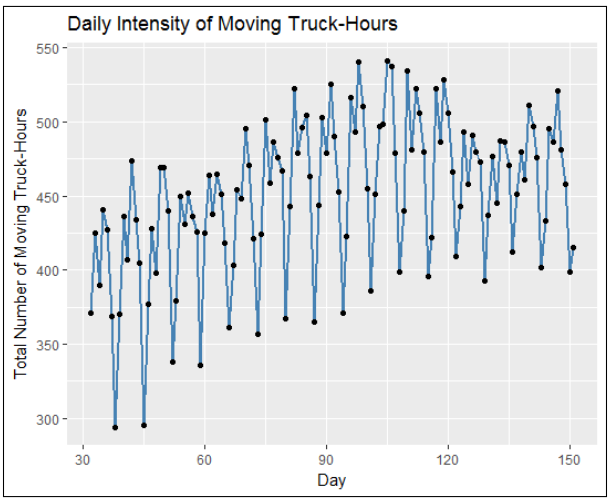

Figure 4.14: Daily intensity of moving truck-hours. The x-axis is the x^{yh} day of the year, and its corresponding y coordinates represent the total number of moving truck-hours.

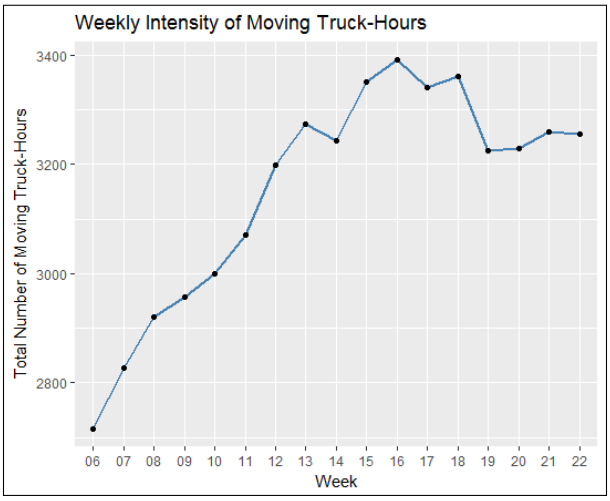

Figure 4.15: Weekly intensity of moving truck-hours. The x-axis is the x^{yh} week of the year, and its corresponding y coordinates represent the total number of moving truck-hours.

can exploit the tools provided in the toolbox to filter the time period to their desired width that suits best to this visualization.

If the time period is longer than a few years, users may want to visualize any monthly pattern in the intensity of moving trucks. The fourth visualization(Figure 4.16) represents such pattern. More specifically, in this visualization, the x-axis is the month of the time period, and its y-axis is the number of truck-hours accumulated over that month.

The fifth visualization((Figure 4.17) contains seven graphs, each corresponding to one day of the week. For example, the curve for Saturday is the average of all Saturdays of our time period. In this visualization, the x-axis and y-axis are respectively the hour of the day and the

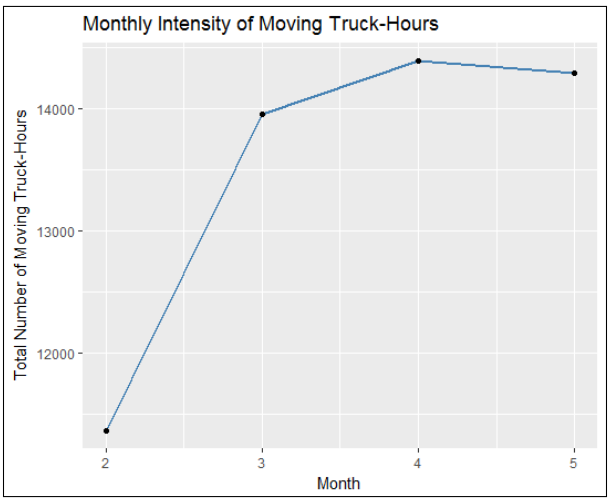

Figure 4.16: Monthly intensity of moving truck-hours. The x-axis is the x^{th} month of the year, and its corresponding y coordinates represent the total number moving truck-hours.

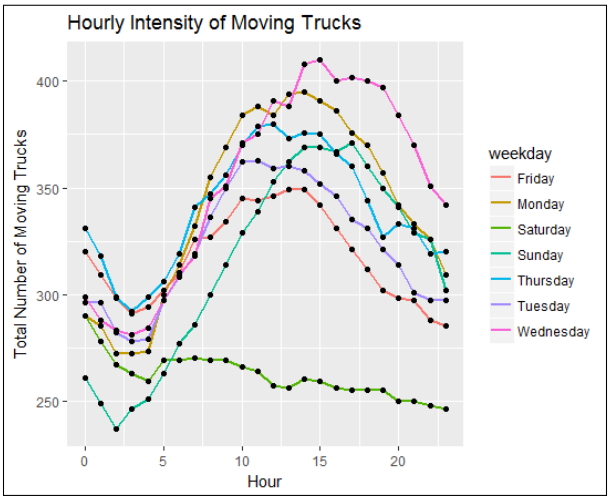

Figure 4.17: Profile of hourly intensity of moving trucks for each day of the week. For example, the y-coordinate of graph Monday corresponding to hour x is the sum, over all Mondays of our trip-base database, of total number of moving trucks during hour x.

total number of moving trucks. For example, the y-coordinate of graph Monday corresponding to hour x is the sum (over all Mondays of our trip-base database) of the total number of moving trucks during hour x. One can compare the intensity curves of weekdays against those of the weekends to determine: which hour has higher intensity during the week, or in the weekends; Or, whether there is any difference between the weekdays in terms of the intensity of moving trucks; Or, whether one day of the week has significantly different intensity curve from another. Another observation from this figure can be the peak hour. For example, it would be clear from this figure if the peak hours of moving intensity are the same among different days of a week.

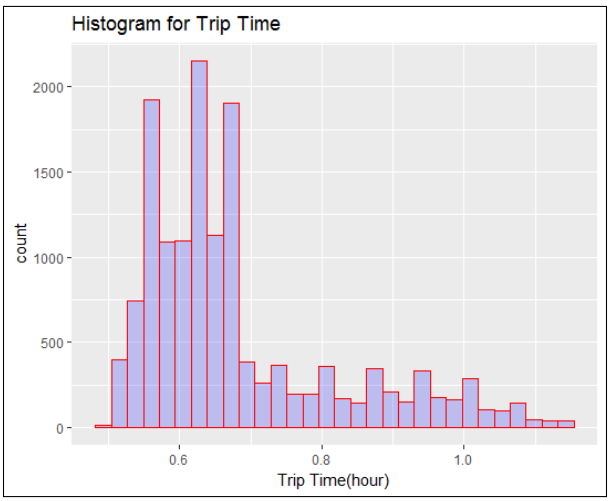

Figure 4.18: Histogram of the trip time.

Trip Visualizations

Thanks to the newly created database, the developed R-based web application will generate two visualizations on this page. The first one is a histogram of the trip time (Figure 4.18). An example of the application of information extracted from this histogram is for the transportation companies to allocate proportionally more drivers to the trips that take longer. The distribution of drivers allocation should be similar to the distribution of trip time; if the number of trips that takes longer is m factor larger/smaller than the ones that take shorter trip time, the number of drivers allocated to such trip should be roughly m times more/less than the number of drivers of shorter trips.

The second one, shown in Figure 4.19, is a histogram of the stop-elapsed-time (the length of a stop between two consecutive trips.) An example of the application of information extracted from this histogram is that this information can be used along with the accident information to educate drivers as to what their stop-elapsed-time profile should optimally be to avoid possible dangerous driving events.

Tools Panel

The tools panel which is located on the left side of the Historical Analysis page consists of six sets of tools which are explained in the following. Each set of tools relates to specific filtering based on some characteristics of data. Once these filtering specifics are set by a user, the

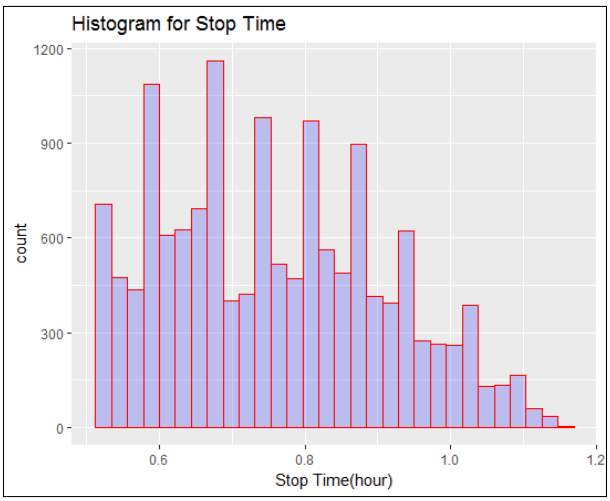

Figure 4.19: histogram of the stop-elapsed-time.

constructed trip-base database that our web application constructs gets immediately modified accordingly such that all visualizations are applied to only those parts of the data that comply with these filtering settings. This is very helpful for users because it enables them to analyze those parts of the data that they are interested in, and if they have any particular interest in some parts of data, it can be done via these filtering (Figure 4.20).

The first set of tools deals with the origin and destination of trips. Four sliders are designed for each of the origin and destination locations. The user can filter the trip-base table by limiting the origin and destination latitude and longitude or shortening the range of date and hours of the trip. This feature allows users to focus on a smaller geographical area and a shorter period of time that is of their interests.

Users can also filter the database based on the day of the week. The second set of tools provides a checkbox for each day of the week. This feature allows users to specify which days of the week to be included in the visualizations. For example, a user may want only a specific day of the week to be included in the visualizations. Or, users may be interested to compare trips over weekends with the ones on weekdays.

The third set of filters enable users to filter the database to only include trips with specific limitations on their trip time and/or stop elapsed time. For instance, a user can filter the trips that have trip duration of less than one hour. Or, to only include those trips with stop elapsed

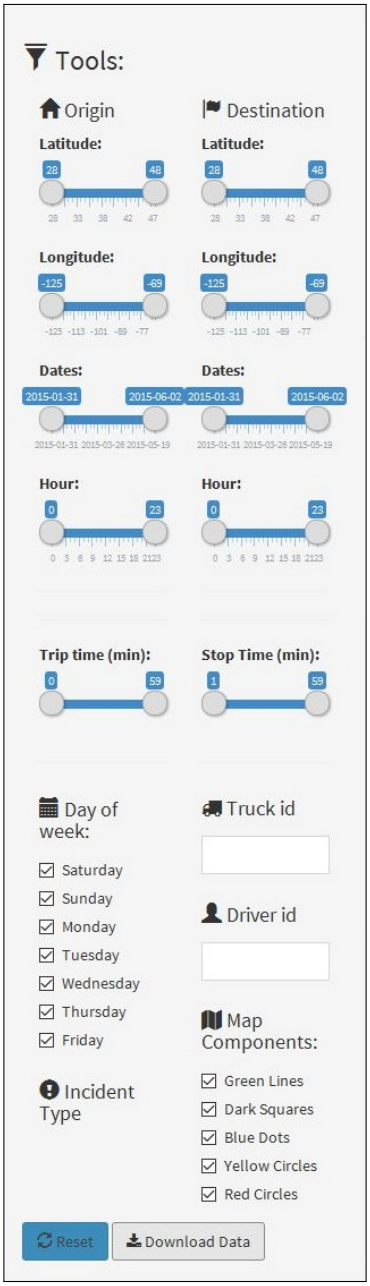

Figure 4.20: The tools panel which is located on the left side of the Historical Analysis page consists of six sets of tools

time of less than half an hour. These filtering can be done by adjusting the two sliders of Trip time and Stop time.

The fourth set of filters apply to Truck and driverIDs. If a user is interested in only analyzing the trips that a specific driver have made or the trips in which a particular truckID has been involved, the user can specify that truckID or driverID in the truckID and driverID sliders. These visualizations and features can be data mined to extract some interesting characteristics

about the trips a specific driver makes, such as whether the trips are mainly inter-states or within a state, are they repeated every day/week or are different.

The fifth set of filters that can be applied to our trip based database allows users to specify the types of incidents they want to include in their visualizations. An example of incident types is hard-break. If users wish to have this incident type excluded or included in their analysis, the web application provides them with this capability.

The last set of filters is only related to the map visualizations. The user can specify what components to appear on the map visualization. The map components are (1) the black squares; (2) the blue dots centroid explaining the centroid locations of stops; (3) the green lines connecting the trips' origin and destination; (4) the yellow circles describing the locations of incidents; and (5) the red circles showing the fatality accidents locations on the map. For each of these five components, there is a checkbox in the tools panel that the user can uncheck to remove that component from the map. The good thing about this feature is that when the size of the trip-base table is large, the user can remove the less necessary components to make the map easier to read.

If after some filtering is performed, users prefer to have the original database for further analyses, the Reset button would allow them to go back to the original database, i.e., the reset button would remove all filtering that the user has specified.

Through Download button, the filtered (or unfiltered) data can be downloaded as a CSV file.

4.3.4 Real-Time Analysis Page

On this page, users can collect and convert GPS locations data in *real-time*. As shown in Figure 4.21, there are five components on this page which are described below.

- **Database drop-down lists.** There are three drop-downs for three databases of GPS, Incidents, and Accidents. Each drop-drown contains all databases of that category (GPS, Incidents or Accident) that the web application has been connected to from the upload page. Users have to choose the database they wish to get real-time data from.

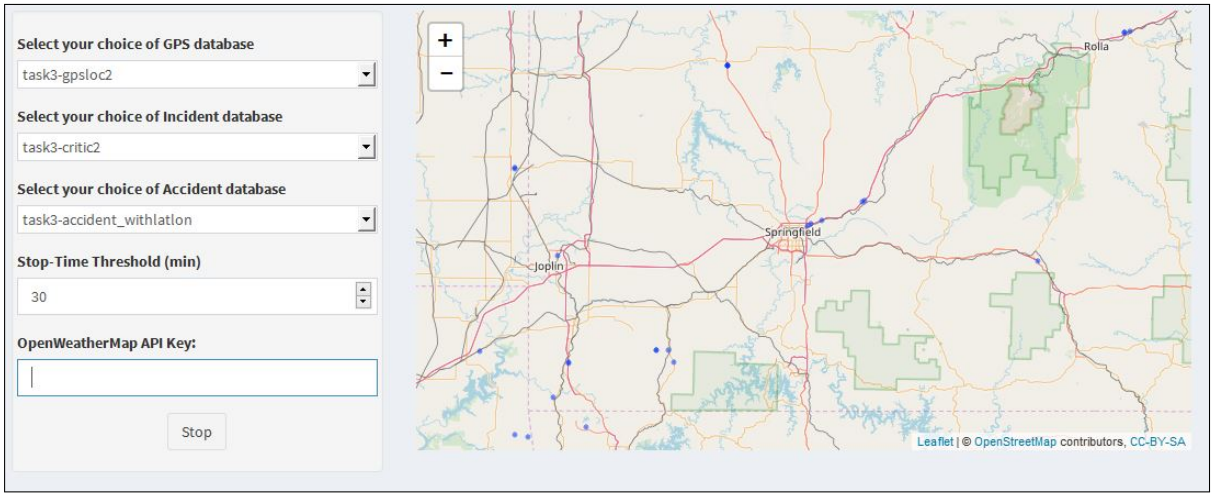

Figure 4.21: A Snapshot of the Real-Time Analysis Page. The blue circles of the map on the right side of the figure are the GPS locations that are updated every minute in real-time.

- **Stop-Time threshold.** In this field, the user can specify the Stop Threshold (in minutes). Stop threshold is explained in Section 4.2.4.

- **Start/Stop button.** Once the database is selected, users can click on the start button. Once the start button is clicked, the web application connects to the selected database and starts reading data in real-time from that database every one minute. Also, every minute, the trip identification algorithm gets applied to all data collected. Once the data conversion (trip identification) at each minute is over, all identified trips get automatically added to the trip-based database, and users can start analyzing them or visualizing them from the Historical Analysis page. In other words, all functionalities of the Historical Analysis page now apply to the newly collected and converted data as well as the previously available data. This data collection and conversion continue until the stop button is clicked.

- **OpenweatherMap API Key field.** Unfortunately, the NOAA database does not include real-time weather information. For the real-time analysis, we use the OpenWeatherMap API [21]. The user needs to enter their own OpenWeatherMap API key in this field. The OpenWeatherMap API key is a unique set of characters that OpenweatherMap provides for each user. Using that API key, each user can request for weather information up

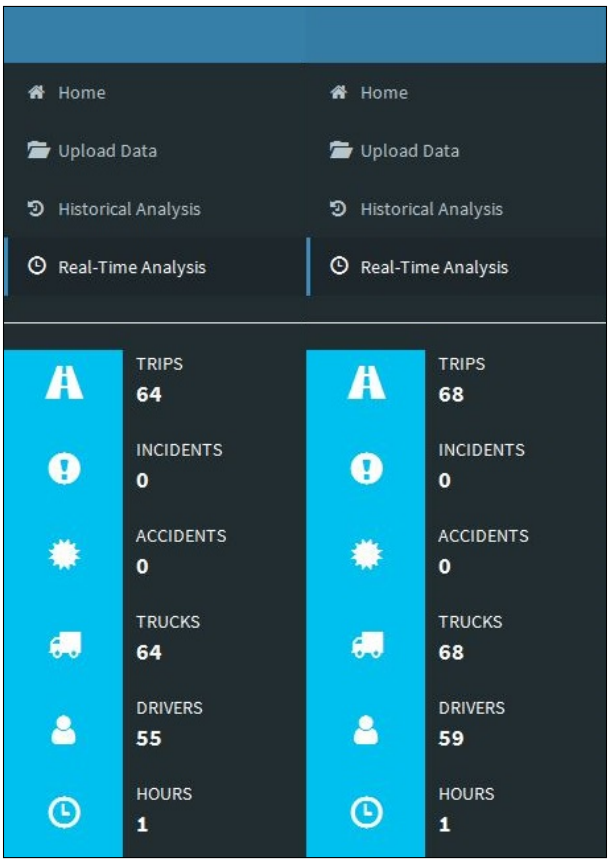

Figure 4.22: Summary box gets updated as new trips are identified in real time. On the right side is a snapshot of the Summary box taken a few minutes after the left one is taken.

to 60 times per minute without any charges. This API key can be obtained from the OpenWeatherMap website [21].

- **Map.** All real-time GPS locations that are read from the database every minute are shown by blue circles on the map.

Figure 4.22 shows two screenshots of the summary box which belong to the real-time analysis on one database but at two different times. The right summary box screenshot is taken a few minutes after the left screenshot was taken. As one can see, the summary information gets updated by time as new trips are identified in real-time

4.4 Results and Discussion

The result of this project is a web application based on an R-based web application and is fully described in Section 4.3.

We run our application for a sample GPS database with 629,770 reported locations. This sample belongs to 54 trucks of a transportation company for a period of one year. The stop threshold was set to 30 minutes. The application identifies 14,638 trips which decrease the number of points by order of 43.

We believe that the output of this application provides brand new information which cannot be seen in the GPS database. For example, a lot of variables regarding the schedules of drivers can be extracted from the trip base database. These variables besides the weather factors might be used by researchers to answer some new research questions about the effects of driving schedules on the probability of an incident or an accident taking place. For example, (1) Can the odds of crash be decreased by having more stops?; (2) Do accidents happen more frequent on long trips than short trips?

4.5 Conclusion and Future Works

We have developed a web application based on an R-based web application (that we have developed) which automates construction of a new trip-base database that addresses the inherent issues of the GPS database, enriching the constructed trip-base database with the weather conditions that a specific trip encountered, enriching the newly created trip-base database with the information of Accidents and Incidents that took place during a particular trip, and generation of several interactive visualizations that describe different characteristics of the trip-base database. This developed web application can be used by researchers for statistical analysis.

One main direction for the future work is to categorize the stops based on the main purpose drivers make a stop at that location. Drivers stop at different locations for different reasons and for different time periods. Some examples are listed as follows:

- Gas Stations: drivers may stop for a few minutes to fill their vehicles tank or/and to buy some snacks for a few minutes.
- Warehouses: drivers may stop for up to a few hours to load/unload cargo.
- Rest Areas: drivers may stop for a few hours to rest or to sleep.

We believe that this type of information can be highly used by researchers and transportation companies in the optimizations analysis. Unfortunately, our web application at this stage is not able to determine stops type and purpose.

As a direction for future work, we propose two approaches for this problem. The first one is to download the OpenStreetMap database and data mine stop category based on its nearby locations and the duration of stops at that location, and then add the type of stop as another variable to our trip-based database. For example, if the OpenStreetMap shows that the stop location is a gas station and the stop duration is around 20 minutes, it is highly likely that this was a stop for filling the vehicles tank and not loading goods.

Since the areas we have defined in our work are very small areas (one-mile-square), it is very unlikely to have more than one type of stop in an area. The second method we propose to find the type of stop is to first categorize all the small areas based on the historical data. From this categorization, one can find the type of stop based on the category of the area the stop belongs to. For example, if we know from the OpenStreetMap that a specific area includes warehouse and we have so many records of stops in this area with time duration of a few hours, we can guess that this is a warehouse that the company mostly uses to load/unload goods. In conclusion, we believe that the OpenStreetMap data, the number and duration of historical stops, and the duration of the current stops can be used to determine the type of stop.

Chapter 5

Conclusion and Summary of Dissertation Contributions

In this dissertation, two real-world problems are addressed. Both problems deal with large-scale data and fall in the broad field of Big Data that has attracted significant attention in the past decade mainly due to the fast-paced advances in the technology. We have developed practical solutions to these two problems by exploiting a variety of reliable yet free databases that different online services provide to the public. We have implemented our solutions in two different web applications.

The first contribution of this dissertation is the development of a web application that finds a fuel efficient speed profile for a given origin and destination locations. The web application takes into account a variety of factors from users' preferences and priorities to weather conditions to road information.

The web application allows users to specify their preferences and priorities by adjusting several parameters on the web. These priorities are very individual dependent and include how much trip's time is important to the driver relative to the importance of the fuel consumed for that trip; how much driving actions, such as the change in speed and taking the brake, the driver is willing to have during the trip. These factors matter significantly to the drivers, especially those who drive frequently.

Further, users have the opportunity to enter the information of their car's engine into the web application so that the optimal speed profile is tailored to their car as well. The optimal speed profile depends to some degrees on the vehicle's information, especially on its mass.

The real-time weather condition is very important when it comes to the safety of driving. It is expected for a good speed profile to be different on a rainy day than on a sunny day. Our speed profile is tailored to the real-time weather conditions.

The developed web application connects to multiple online databases such as GoogleMaps, OpenStreetMap, and OpenWeatherMap databases to data mine all required information and then solves an optimization problem using a dynamic programming model.

We have used our web application for a variety of origin and destination locations. The results show a good range of fuel consumption saving of up to 15%. The saving highly depends on the user's preference, weather, and road conditions.

Further to its practicality for individual users, our application outputs a lot of information about the output speed profile. This information is shown on the web application via different graphs. The detailed data of the output speed profile, brake profile, consumed gas profile, and gear profile can be downloaded to be used by researchers for further data analysis. There are two main directions for future work on this problem. The first one is to increase the speed of the web application and reduce the amount of memory it requires. There are two main bottlenecks for the speed of our web application which are the dynamic programming model and the road identification. Using more efficient algorithms and parallel computing might decrease the time of the web application. The other option is to study and analyze faster (than dynamic programming) existing methods to solve the optimization problem.

The second direction of future work relates to the accuracy of the algorithm used to identify roads in the OpenStreetMap database. Speed limits depend on the identified road, so is the location of stop signs, traffic signals, and yield signs. Therefore, it is critical for the road identification algorithm to be accurate. However, improved accuracy of such algorithm does have a reverse relation with its speed. Therefore, it is critical to study and develop more advanced road identification algorithms that have high accuracy but are not slow.

The third direction of future work is to take into account more real-time information. Real-time traffic flow, real-time incidents, and real-time accidents are three databases that can significantly advance the DRIIVE application.

The forth direction of future work is to add fuel consumption models of other types of vehicles such as electric vehicles, hybrid electric vehicles, etc.

The second contribution of this dissertation is the development of an R-based web application along with a web application that automates the conversion of a raw GPS locations

database into a trip-based database that readily contains semantic attributes about trips and stop locations.

Users can upload a GPS locations database, an accident database, and an incident database on the web application. Upon the completion of the upload, a trip-based database gets created, each row of which is a unique trip with its origin and destination locations, its start and end time, the duration of the stop immediately following it, the weather information the trip was facing, and any accident or incident the vehicle was involved in during that trip.

This database can be downloaded from the web application for further research and analysis by researchers. In addition, multiple visualizations that illustrate different characteristics of the identified trips are provided by the web applications. The web application is also able to apply the aforementioned functionalities on a GPS locations database that gets updated in the real-time.

Two main directions of future work for this contribution are as follows. The first one is to develop the trip identification algorithm further. Trip identification has extensively been studied in a very general form in the literature. However, more sophisticated algorithms that are tailored to our particular application need to be developed.

The second direction of the future work is to include road conditions to the database. The road conditions can be collected from the free database of OpenStreetMap. For example, the speed limit, number of traffic lights, and number of lanes of the roads in a trip are the variables that may help researchers build more powerful crash prediction models.

References

[1] Leaflet for R. https://rstudio.github.io/leaflet/.

[2] Erik Hellström. Explicit use of road topography for model predictive cruise control in heavy trucks, 2005.

[3] Co2 emissions from fuel combustion by sector in 2014. http://www.iea.org/publications/freepublications/publication/co2-emissions-from-fuel-combustion-highlights-2016.html.

[4] Kyoungho Ahn, Hesham Rakha, Antonio Trani, and Michel Van Aerde. Estimating vehicle fuel consumption and emissions based on instantaneous speed and acceleration levels. *Journal of transportation engineering*, 128(2):182–190, 2002.

[5] Min Zhou, Hui Jin, and Wenshuo Wang. A review of vehicle fuel consumption models to evaluate eco-driving and eco-routing. *Transportation Research Part D: Transport and Environment*, 49:203–218, 2016.

[6] Michael Sivak and Brandon Schoettle. Eco-driving: Strategic, tactical, and operational decisions of the driver that influence vehicle fuel economy. *Transport Policy*, 22:96–99, 2012.

[7] Hans Jakob Walnum and Morten Simonsen. Does driving behavior matter? an analysis of fuel consumption data from heavy-duty trucks. *Transportation research part D: transport and environment*, 36:107–120, 2015.

[8] Rich C McIlroy and Neville A Stanton. What do people know about eco-driving? *Ergonomics*, 60(6):754–769, 2017.

[9] Toshihiro Hiraoka, Seimei Nishikawa, and Hiroshi Kawakami. Driver-assistance system to encourage spontaneous eco-driving behavior. In *Proceedings of 18th World Congress on Intelligent Transport Systems, CD-ROM*, 2011.

[10] Bart Beusen, Steven Broekx, Tobias Denys, Carolien Beckx, Bart Degraeuwe, Maarten Gijsbers, Kristof Scheepers, Leen Govaerts, Rudi Torfs, and Luc Int Panis. Using on-board logging devices to study the longer-term impact of an eco-driving course. *Transportation research part D: transport and environment*, 14(7):514–520, 2009.

[11] Jack N Barkenbus. Eco-driving: An overlooked climate change initiative. *Energy Policy*, 38(2):762–769, 2010.

[12] Thijs van Keulen, Bram de Jager, Darren Foster, and Maarten Steinbuch. Velocity trajectory optimization in hybrid electric trucks. In *American Control Conference (ACC), 2010*, pages 5074–5079. IEEE, 2010.

[13] AP Stoicescu. On fuel–optimal velocity control of a motor vehicle. *International Journal of Vehicle Design*, 16(2-3):229–256, 1995.

[14] Benjamin Passenberg, Peter Kock, and Olaf Stursberg. Combined time and fuel optimal driving of trucks based on a hybrid model. In *Control Conference (ECC), 2009 European*, pages 4955–4960. IEEE, 2009.

[15] Wei Huang, David M Bevly, Xiaopeng Li, and Steve Schnick. 3d road geometry based optimal truck fuel economy. In *ASME 2007 International Mechanical Engineering Congress and Exposition*, pages 63–70. American Society of Mechanical Engineers, 2007.

[16] Erik Hellström, Maria Ivarsson, Jan Åslund, and Lars Nielsen. Look-ahead control for heavy trucks to minimize trip time and fuel consumption. *Control Engineering Practice*, 17(2):245–254, 2009.

[17] Erik Hellström, Jan Åslund, and Lars Nielsen. Design of an efficient algorithm for fuel-optimal look-ahead control. *Control Engineering Practice*, 18(11):1318–1327, 2010.

[18] Google Maps javascript api. https://developers.google.com/maps.

[19] Karl Johan Astrom and Lars Rundqwist. Integrator windup and how to avoid it. In *American Control Conference, 1989*, pages 1693–1698. IEEE, 1989.

[20] Melonie P Heron. Deaths: leading causes for 2011. 64:1–96, 2015.

[21] Openweathermap api. https://openweathermap.org/api.

[22] MySQL. https://www.mysql.com/.

[23] Django web framework. https://www.djangoproject.com/.

[24] Responsivevoice instant text to speech javascript and speech synthesis library. https://responsivevoice.org/.

[25] amCharts javascript charts and maps. https://www.amcharts.com/.

[26] Matplotlib python plotting. https://matplotlib.org/.

[27] Engin Ozatay, Umit Ozguner, John Michelini, and Dimitar Filev. Analytical solution to the minimum energy consumption based velocity profile optimization problem with variable road grade. *IFAC Proceedings Volumes*, 47(3):7541 – 7546, 2014.

[28] Ondrej Santin, Jaroslav Pekar, Jaroslav Beran, Anthony D'Amato, Engin Ozatay, John Michelini, Steven Szwabowski, and Dimitar Filev. Cruise controller with fuel optimization based on adaptive nonlinear predictive control. *SAE Int. J. Passeng. Cars Electron. Electr. Syst.*, 9:262–274, 2016.

[29] Benjamin Passenberg, Peter Kock, and Olaf Stursberg. Combined time and fuel optimal driving of trucks based on a hybrid model. In *2009 European Control Conference (ECC)*, pages 4955–4960, 2009.

[30] Md Abdus Samad Kamal, Masakazu Mukai, Junichi Murata, and Taketoshi Kawabe. Model predictive control of vehicles on urban roads for improved fuel economy. *IEEE Transactions on Control Systems Technology*, 21(3):831–841, May 2013.

[31] Lei Gong, Hitomi Sato, Toshiyuki Yamamoto, Tomio Miwa, and Takayuki Morikawa. Identification of activity stop locations in gps trajectories by density-based clustering method combined with support vector machines. *Journal of Modern Transportation*, 23:202–213, 2015.

[32] Jorge Alvarez-Lozano, J. Antonio Garca-Macas, and Edgar Chvez. Crowd location forecasting at points of interest. *International Journal of Ad Hoc and Ubiquitous Computing*, 18:191–204, 2015.

[33] Cornelia Schneider, Simon Grochenig, Verena Venek, and Michael Leitner. A framework for evaluating stay detection approaches. *International Journal of Geo-Information*, pages 1–19, 2017.

[34] Lei Zhu and D. Jeffrey Gonder. A driving cycle detection approach using map service api. *International Journal of Ad Hoc and Ubiquitous Computing*, 92:349–363, 2018.

[35] Glenn Cich, Luk Knapen, Tom Bellemans, Davy Janssens, and Geert Wets. Trip/stop detection in gps traces to feed prompted recall survey. *International Journal of Ad Hoc and Ubiquitous Computing*, 52:262–269, 2015.

[36] Sivaramnakrishnan Srinivasan, Stacey Bricka, and Chandra Bhat. Methodology for converting gps navigational streams to the travel-diary data format, 2009.

[37] Luis Otavio Alvares, Vania Bogorny, Bart Kuijpers, Jose Antonio Fernandes de Macedo, Bart Moelans, and Alejandro Vaisman. A model for enriching trajectories with semantic geographical information. In *Proceedings of the 15th Annual ACM International Symposium on Advances in Geographic Information Systems*, pages 22:1–22:8. ACM, 2007.

[38] Peter Stopher, Camden FitzGerald, and Jun Zhang. Search for a global positioning system device to measure person travel. *Transportation Research Part C: Emerging Technologies*, 16:350–369, 2008.

[39] J. Wolf, Stefan Schonfelder, Ute Samaga, M. Oliveira, and Kay W. Axhausen. Eighty weeks of global positioning system traces: Approaches to enriching trip information. *Transportation Research Record: Journal of the Transportation Research Board*, 1870:46–54, 2004.

[40] Nadine Schuessler and Kay Axhausen. Processing raw data from global positioning systems without additional information. *Transportation Research Record: Journal of the Transportation Research Board*, 2105:28–36, 2009.

[41] Jean Wolf, Randall Guensler, and William Bachman. Elimination of the travel diary: Experiment to derive trip purpose from global positioning system travel data. *Transportation Research Record: Journal of the Transportation Research Board*, 1768:125–134, 2001.

[42] Xin Cao, Gao Cong, and Christian S. Jensen. Mining significant semantic locations from gps data. *Proc. VLDB Endow.*, 3:1009–1020, 2010.

[43] Ilkcan Keles, Christian S. Jensen, and Simonas Saltenis. Extracting rankings for spatial keyword queries from gps data. pages 173–194. Springer International Publishing, 2018.

[44] NOAA weather data. ftp://ftp.ncdc.noaa.gov/pub/data/noaa/.

[45] Dederal climate complex data documentation for integrated surface data. ftp://ftp.ncdc.noaa.gov/pub/data/noaa/ish-format-document.pdf.

[46] FARS fatality analysis reporting system. ftp://ftp.nhtsa.dot.gov/fars/.

[47] Shiny an R Package. http://shiny.rstudio.com/.

[48] Date Formats in R. https://www.r-bloggers.com/date-formats-in-r/.