Leveraging Smartphones to Incentivize City-Wide, Energy Efficient Transportation

by

Arjun Srinivasan

Submitted to the Department of Electrical Engineering and Computer Science on February 3, 2017, in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

Transportation is a major source of energy consumption in developed countries [4]. Unfortunately, people have little incentive to change their habits, relying on inefficient transportation sources. The "Sustainable Travel Incentives with Prediction, Optimization and Personalization" (Tripod) project seeks to incentivize people to improve their transportation-related behavior through redeemable tokens. These tokens are rewarded through city-wide, energy-optimized transportation decisions decided in real-time. As part of this initiative, I have developed an initial prototype Android application named FMS Advisor that allows users to interact with the larger Tripod system before, during, and after their journey.

Working off an initial design, I designed a trip planner interface that uses optimized route planning information to display a personalized trip menu. I also developed trip validation algorithms such as vehicle occupancy detection and driving-style detection using dynamic time warping (DTW) and threshold-based methods. These methods were then evaluated through group user sessions and controlled trip experiments. This resulted in a functional end-to-end user experience, though trip validation methods require additional data to have properly tuned detection. In the future, token redemption will be possible through an integrated marketplace for rewards that can be accessed at the end of a trip.

Thesis Supervisor: Moshe E. Ben-Akiva
Title: Edmund K. Turner Professor of Civil and Environmental Engineering
Acknowledgments

I would like to thank the ARPA-E Transnet program from the Department of Energy (DoE) under the funding agreement DE-AR0000611 for funding this project. I would also like to thank Carlos Azevedo, Pablo Ortiz, and Bilge Atasoy for providing guidance throughout the project and reviewing my thesis work.
## Contents

1 Introduction 13

2 Background 17
   2.1 Traffic Congestion Solutions ................................. 17
   2.2 Trip Planning Interfaces .............................. 18
   2.3 Driving-Style Incentives ................................. 19

3 Design 21
   3.1 Overview ............................................. 21
   3.2 Android Application ..................................... 22
      3.2.1 Trip Planner ..................................... 24
         3.2.1.1 Route Selection .............................. 24
         3.2.1.2 Trip Completion ............................. 28
      3.2.2 Trip Validation .................................... 29
         3.2.2.1 Core FMS Validation Methods ......................... 29
         3.2.2.2 Driving Style Detection - Overview ................. 30
         3.2.2.3 Driving Style Detection - Dynamic Time Warping ... 30
         3.2.2.4 Driving Style Detection - Threshold-based Methods ... 31
         3.2.2.5 Data Logging and Collection ....................... 32
         3.2.2.6 Vehicle Occupancy Detection ...................... 33
      3.2.3 Marketplace ....................................... 34
   3.3 User Preferences Database ................................... 34
   3.4 System Optimization and Fallback Route Determination .... 35
3.5 User Optimization ............................................ 36

4 Evaluation ......................................................... 37
   4.1 Methodology ............................................... 37
   4.2 Component Testing ......................................... 38
      4.2.1 Experimental Setup ................................. 38
      4.2.2 UI Testing .......................................... 38
      4.2.3 Backend Testing .................................... 40
      4.2.4 DTW Driving Style Detection Testing .......... 42
      4.2.5 Threshold-based Driving Style Detection Testing . 44
      4.2.6 User Optimization Performance .................. 46
      4.2.7 Future Testing Strategy ............................. 47

5 Conclusion ......................................................... 49
List of Figures

1-1 FMS Activity Diary ........................................ 14
1-2 FMS Advisor Initial Design .................................. 16

3-1 Architecture ................................................... 22
3-2 Flowchart ...................................................... 24
3-3 Navigation drawer .............................................. 25
3-4 Bluetooth scans ............................................... 26
3-5 Places API ...................................................... 26
3-6 Route list ....................................................... 27
3-7 Google Maps navigation ....................................... 27
3-8 Trip-in-progress ............................................... 28
3-9 Trip completion ................................................ 29
3-10 Accelerometer thresholds .................................... 32

4-1 Bug reporting survey .......................................... 39
4-2 Bug reporting breakdown ..................................... 40
4-3 Route for backend testing ................................... 41
4-4 Unfiltered accelerometer sensor data .................... 43
4-5 Filtered accelerometer sensor data ......................... 43
4-6 OBD map ......................................................... 45
4-7 Accelerometer threshold data ................................ 46
4-8 Traceview profiling ............................................ 47
List of Tables

4.1 Backend server stats .............................................. 41
4.2 Hard acceleration OBD data ...................................... 44
4.3 Hard acceleration FMS data ...................................... 46
Chapter 1

Introduction

The Tripod project is a large project focused on personalised incentives targeting system-wide optimized energy savings. To achieve this goal, the Tripod team has designed a framework with multiple components. A core component of the project is the user facing component, a mobile application (app) for use by potential users of this system. The app would track user behavior and interface with the larger Tripod projects backend components. An existing application, Future Mobility Sensing (FMS) [32, 21], is a travel survey app that can perform mode detection, stop detection, and route validation with the use of a central server. Mode detection allows the application to determine the mode of transportation through the use of inbuilt smartphone sensors, while stop detection determines changes between different modes of transportation by separating different legs of the trip using time periods with limited movement. Route validation verifies that the correct route was recorded by FMS. The Android application that I worked on was implemented as a new application based on the FMS project, called FMS Advisor.

DynaMIT [17] is a project under the Tripod umbrella that simulates and predicts traffic and public transportation conditions. By leveraging data from various sources, such as traffic and toll data, DynaMIT seeks to guide users toward optimal decisions. TripEnergy [27] is an energy and emissions model that predicts changes based on travel. DynaMIT, combined with the energy model TripEnergy, will conduct simulations over the Greater Boston area, that can be leveraged by FMSAdvisor. The
DynaMIT framework will allow computing the optimum energy savings per tokens to be provided at every time step. FMS Advisor will utilize the predictive information and energy-based optimized incentives from this system-wide optimization. In the future, the project hopes to move on to simulate other areas.

Future Mobility Sensing (FMS) is a travel survey app that has a few key pieces of functionality. The major goal of this app is to track the travel activities of users throughout the day, and offer a reconstruction of the trips taken by the user from distinct locations. To do this, smartphone sensors such as the accelerometer, GPS, and Wi-Fi are utilized to create a database of information that can be further analyzed. Algorithms to perform stop detection, mode detection, and other forms of trip validation using the sensor data then aid in the creation of a complete travel diary.

![Sample activity diary from the FMS web interface.](image)

The application has account functionality, which allows the travel log (Figure 1-1) to be viewed on a desktop web browser for easier visualization. This account
functionality allows travel logs to be associated with particular users, and to be stored in a central backend database for more sophisticated data analysis. In addition, this allows for a web interface for better visualization of travel data. Sensor data, such as that from GPS, Wi-Fi, accelerometers, or barometers, is also sent to the backend server for additional data analysis by related projects.

FMS Advisor is a new application within the framework of the core FMS app. Taking the passive logging functionality and trip tracking, FMS Advisor provides additional active capabilities that allow trips to be explicitly planned and routed using simulation data of the Tripod project. This is done through the addition of multiple elements: a direct user interface for trip planning and navigation, backend processes for enhanced trip validation, and a planned incentive system. The purpose of FMS Advisor is to leverage the work of the Tripod project in global energy efficiency simulation to motivate users directly using incentives.

The Tripod project as a whole seeks to combine these elements to allow the user to make energy-conscious transportation decisions. The user can request a trip, and be provided with a personalized menu of routes. Alternative routes are provided based on user preferences and real-time data. These include alternate modes of transportation such as cars, bicycles, public transportation, and services such as Uber or Lyft. These routes have associated token values, which can be exchanged in a marketplace for rewards. These token values are determined by a variety of factors, including mode of transportation, departure time, route choice, driving style, and the choice of carpooling. After the user selects a route, turn-by-turn navigation is launched and allows the user to navigate to their intended destination. While this is happening, sensor data logging and trip validation occurs to ensure the user follows the selected route. After the completion of a trip, the token values promised at the start of the trip are awarded.

My contribution in the Tripod project is in the development of FMS Advisor and its integration with the other components of the Tripod project. Specifically, I worked on the design and development of the trip planner interface based on an initial design, as shown in Figure 1-2. This work utilized Google APIs for directions and
turn-by-turn navigation, and involved preliminary integration of personalized ranking of routes. I also worked on validation algorithms such as vehicle occupancy detection and driving style detection and integration with backend servers. I was responsible for initial testing of these features, though further testing still remains to be done. In addition, aspects such as the marketplace component are not yet complete.

Figure 1-2: Initial design of FMS Advisor interface.
Chapter 2

Background

2.1 Traffic Congestion Solutions

Transportation accounts for a large amount of energy consumption in developed countries [4]. Current methods of solving this problem have focused on fuel efficiency [2], but other factors like traffic congestion and planning of energy efficient routes can affect energy consumption. The Tripod (“Sustainable Travel Incentives with Prediction, Optimization and Personalization”) project was created to address these concerns. The project seeks to build a centralized system to reward smartphone users tokens as incentives for using energy efficient transportation.

Incentives are useful in motivating users to take routes that are otherwise sub-optimal in time, distance, or cost. They may also help in the reduction of traffic congestion [18], though earlier efforts to use them focused on fixed incentives based on pre-decided transportation alternatives. Tripod is the first project to operationalize incentive-based transportation systems on a large scale, optimized for energy-efficiency and in real-time.

Prior studies have attempted to address transportation issues in various ways. The Electronic Road Pricing (ERP) [11] system in Singapore was the first to utilize variable pricing for congestion controls. It used short-range radio communications to collect tolls in heavily congested areas. A fixed quota of vehicles is also used in Singapore through the Certificate of Entitlement (COE) [8] system, which only allows
a fixed number of vehicles to be driven in Singapore. Both of these methods rely on physical infrastructure to enforce their congestion controls.

An infrastructure-free solution, Roadrunner [22], addressed the issue of vehicular congestion in cities by controlling vehicles use of specific road segments via digital tokens. By limiting the number of tokens available to a road network to some fixed quota, the system could effectively ration out road space. The system leveraged short-range, device-to-device communication between smartphones, called dedicated short-range communications (DSRC), to allow vehicles to exchange their relative positions and, more importantly, tokens. In addition to smartphones in other vehicles, each smartphone also communicated with a central server. This server governed the allocation and distribution of tokens at the granularity of an entire road network. Though RoadRunner did not attempt to motivate behavior via the distribution of tokens, the idea of managing travelers in this way could be translated into a general, token-based incentive structure.

Another app, Metropia [1], incentivizes commuters to choose departure times, routes, and modes that help reduce and eliminate traffic congestion via a social rewards-based ecosystem for agencies and companies. These incentive programs have become more popular. The startup RideAmigos [3] created a service that makes use of such a structure. It has features such as trip planning, trip logging, and ridesharing. The service also seeks to improve the adoption of these features by imposing challenges, believing that competition will increase user engagement, but without real-time adjustments to the observed or predicted conditions on the transportation system.

\section{Trip Planning Interfaces}

For actual trip planning, there are several different interfaces and routing software packages in use today. The most commonly used one is Google Maps [12], available across all major smartphone platforms and through a web browser. This allows for routing along multiple modes of transportation (and combinations of them), with
options such as avoiding highways or tolls. In addition, it includes a turn-by-turn navigation interface for use in a driving scenario, which is able to operated hands-free. Google Maps provides several APIs for route planning and for launching the turn-by-turn navigation with specific options, as well as embeddable interactive maps for smartphone applications. Unfortunately, the individual components of the application, such as the trip menu interface, cannot be leveraged separately from the application itself.

Other navigation applications include Apples iOS Maps [6] and Waze [16], which offer similar features across their particular platforms. There also exist open source solutions with freely available map data and planning software, such as OpenTrip-Planner [25], which provide transportation analysis and routing software that can be leveraged by smartphone applications. These make use of a full client-server model that allows requests from a backend routing server through a REST API.

Applications such as Citymapper [9] and Moovit [14] are smartphone applications that allow planning trips using public transit. These leverage bus arrival times, train schedules, and service announcements in their routing algorithms.

### 2.3 Driving-Style Incentives

There have also been previous efforts in areas of trip validation and incentivizing energy-efficient driving. Driving-style detection, which involves monitoring sensor data to identify poor driving, is a major element of trip validation. Previous efforts, such as Johnson and Trivedi [23], have focused on using smartphone sensors to identify aggressive driving. Individual aggressions, such as aggressive turns or braking, are found by comparing incoming sensor data to previously documented templates using dynamic time warping (DTW). This dynamic time warping technique, taken from time-series analysis, allows for infractions to be gleaned from large-scale streaming sensor data. This is done by comparing incoming signal data to predetermined template signals to see if they match particular driving-style events. Incoming data is filtered using a low-pass filter (such as a Butterworth filter [20]) to remove noise.
Many of these aggressive actions, which are definitely negative from a safety standpoint, can be further applied to poor energy efficiency of the trip.

Other previous works use more sophisticated hardware (such as Joubert, Beer and Koker [24]), including onboard cameras, which allow for more complex computer vision techniques to be used to determine aggressive driving. In addition, the accuracy of the hardware allows for more sophisticated machine learning techniques. Unfortunately, given the constraints of the smartphone hardware owned by the average driver, this seems unlikely to be usable in practice for the purposes of the FMS application.

A simpler method of driving-style detection involves the use of sensor thresholds [31]. This involves calibration of the initial orientation of the smartphone, and using empirically determined thresholds on accelerometer data to distinguish particular driving-style events.

One application developed for the city of Boston is for the “Boston's Safest Driver Competition” [5], which provides feedback to users based on various driving metrics. The application focuses on speed, acceleration, braking, cornering, and phone distraction, and scores users across various modes of transportation. In addition to the scoring, a major aspect is the potential for a monetary grand prize.
Chapter 3

Design

3.1 Overview

FMS Advisor builds on top of the existing FMS codebase, utilizing additional interfaces and backend servers. The existing FMS code base consists of a smartphone application (for both Android and iOS) and a Rails-based back-end server. The primary goal of the FMS app thus far has been to collect transportation-related data in a passive manner from its users over the course of many days and require users to recall explicit trips taken through the use of website surveys. FMS Advisor seeks to change the behavior of users by allowing them to explicitly request trips and receive energy-optimized potential routes. FMS Advisor is a new smartphone application that communicates with two additional backend services: the System Optimization (SO) and the User Optimization (UO). These three components (the client-side smartphone app, the server-side SO, and the UO) form the backbone of the Tripod project.

FMS Advisor provides an interface for the user, accepting route parameters, and sends that information to the SO to provide routes with predictive attributes and token energy values. The System Optimization (SO) server makes use of simulated energy models to provide an optimal list of routes given an origin and destination. The SO then communicates that information to the UO, which customizes the ranking of the routes based on user preferences. The UO then relays this information back to FMS Advisor, which allows selection of the preferred route by the user, and begins
Figure 3-1: The workflow that defines the interactions between different components in the system.

turn-by-turn navigation. FMS Advisor will also log data from smartphone sensors for further analysis during the route, which is sent to the backend data server after the route is completed.

3.2 Android Application

FMS Advisor builds upon the original FMS app with additional interfaces and background services to assist on explicit trip planning and validation. There are three primary components of the Advisor application (Figure 3-1): the Trip Planner, enhanced Trip Validation and a token-based Marketplace. A central Dashboard links all of these components together in the application. As of now, the Trip Planner has been built and the work in progress Trip Validation component has reached testing phases, while the Dashboard and Marketplace are still works in progress.

The Dashboard will be the primary form of navigation for the app, providing the user easy access to various user interactions. The Trip Planner and Marketplace interactions can be directly accessed from the dashboard, while Trip Validation is
exclusively a background process. In addition, there will be a status display of current tokens earned and a map of the users nearby area. This main dashboard can be accessed through a sidebar navigation menu, as can all of the other features in the FMS Advisor app.

The Trip Planner uses routes suggested by a backend server to show token-reward sorted options for the user to select. It then allows navigation along the selected route. Trip Validation is then used to verify that trips have been completed as planned and to determine what, if any, token reward will have been earned by a user. Sensors built into each users device will be used to check on the progress of each trip. This Trip Validation includes: mode detection, route validation, and stop detection. Two new sensing modes were added to FMS: vehicle occupancy detection and driving style detection. Vehicle occupancy detection determines other FMS Advisor users in the same vehicle as the primary user, allowing for additional rewards for ridesharing. Driving style detection is done to encourage safe and energy-efficient driving through the detection of energy-inefficient driving behaviors.

The Marketplace interface will provide users with access to the Token Rewards Market. What it displays will depend largely on what data is available to the app. These elements will be integrated into the existing FMS Advisor app, unified through a central Dashboard that provides access to the two key user interactions: planning a trip and exchanging tokens on the Token Rewards Market.

FMS Advisor is built solely as an Android application. The reason for this is twofold: Android currently has the largest user base [19] in the smartphone market, and development resources behind the FMS Advisor project are limited. For this reason, the additions to the FMS application do not currently apply to the iOS version. In the future, such a version might be developed, and would be modeled off of the existing Android-based FMS Advisor.

I worked on developing the trip planner interface, the vehicle occupancy detection, and the driving-style detection components of FMS Advisor. I also assisted with backend data server integration, UO integration and profiling, and expansion of the logging functionality.
3.2.1 Trip Planner

3.2.1.1 Route Selection

The Trip Planner leverages a backend server to aid the user in selecting trip plans. This is done by providing parameters and requesting a list of routes for those parameters. These include options such as origin, destination, mode of transportation, and start time. The final version of FMS Advisor will leverage the SO module on a backend server to provide a list of energy efficient routes with these parameters. While the SO is in development, a fallback option to use Google Maps to search for directions is used.

In the future, the UO module will calculate token values for each suggested route from the SO using energy values. This data will be transmitted to FMS Advisor and presented to the user whenever a new trip is planned. After a route is chosen, the UI will notify the user of next steps similar to turn-by-turn navigation apps by opening an instance of Google Maps navigation. At the end of the trip, the user can return to the FMS Advisor app to see the final token reward, which might change depending on route changes during the navigation.

![Figure 3-2: The workflow of FMS Advisor trip planning.](image)

The workflow (Figure 3-2) of the Trip Planner involves a simple series of steps. The Trip Planner is selected from a navigation drawer (Figure 3-3), upon which the parameter selection screen is displayed. Before any parameters can be input, vehicle occupancy detection (see 3.2.2.6) will take place to detect nearby FMS Advisor users. This involves Bluetooth scans (Figure 3-4), so temporary permissions are requested of the user before the scan takes place. Android requires explicit permissions for the
application to leverage hardware sensors and features, and these must be approved by the user. Permissions can be granted on either a temporary (time-limited) or a permanent per-app basis. Since vehicle occupancy detection is only needed for a small period of time before a trip in its current implementation, a time period of 120 seconds is requested by FMS Advisor for Bluetooth permissions. Section 3.2.2.6 has further details.

Then, the user is asked to input the previously mentioned parameters, including origin and destination. Origin and destination selection utilizes the Google Places API (Figure 3-5), which allows for smart contextual matching of search terms to unique places. This involves using features of locations such as name and address. The mode of transportation selected is the preferred mode for the route, which is taken into account by the UO algorithm.

After directions are requested, communication with the UO server will allow a set of routes (Figure 3-6) to be determined and ranked according to different user preference parameters. The user is then shown a list of routes and can select the route of their choosing. In the future, these lists will have the ability to be sorted or filtered by chosen parameters, such as distance, time, or token value reward.
Google Maps Navigation (Figure 3-7) is then called via an Intent, launching it as a separate application to allow turn-by-turn navigation. Intents are Android-specific methods of communication between different components of an application or different applications entirely. Parameters are packaged into a message format and allow for the intent-caller to obtain results from the message receiver after receiver is
Figure 3-6: Routes displayed to the user after communication with the SO and UO.

done processing the message.

Figure 3-7: Google Maps navigation using FMS Advisor-provided waypoints.

Google Maps Navigation is requested using a sequence of waypoints. These way-
points are determined by breaking down the route selected by the user into sets of
coordinates along portions of the route. The route is then reconstructed by the Google
Maps application as a sequence of origin to destination pairs, allowing the specific route to be navigated through the Google Maps Navigation interface.

### 3.2.1.2 Trip Completion

During the turn-by-turn navigation, the trip planner interface is inactive, displaying a message that a trip is currently in progress (Figure 3-8). This prevents a user from accidentally requesting multiple trips at once, and makes sure that logged sensor data is properly sent to the data server at the end of the trip.

![Trip Planner Interface](image)

**Figure 3-8:** Trip-in-progress message during turn-by-turn navigation.

Trip completion (Figure 3-9) is determined through the use of geofences. Geofences are an Android-specific method of tracking location in the background and notifying an application if a certain geographical region has been reached. This is done by marking out a radial area from a specific latitude/longitude coordinate pair, called a geofence. Upon entering the geofence, a message is sent to the main application from the background service, and the trip ends with a trip completion message. The message will contain future information when marketplace-related features are fully developed.

FMS Advisor relies heavily on notifications to provide feedback to the user, es-
pecially when the application is not in the foreground. Androids notification system allows applications in the background to deliver messages that can also be interacted with to perform specific actions. FMS Advisor uses notifications during the turn-by-turn navigation to inform the user about aggressive driving as well as the end of the trip. The end of trip notification allows the user to return to the FMS Advisor app from Google Maps.

3.2.2 Trip Validation

Trip validation concerns all of the background services running during a trip that exist to verify that the trip was completely successfully and correctly. It consists of a few major components: mode detection, route detection, stop detection, driving style detection, and vehicle occupancy detection. The first three components leverage core FMS, while the other two pieces are new to FMS Advisor.

3.2.2.1 Core FMS Validation Methods

Core FMS [32] contains three different trip validation methods: mode detection, route detection, and stop detection. Mode detection determines what mode of transporta-
tion is taken by the user. Mode detection uses machine learning on data taken from
the accelerometer and GPS to identify mode. Route detection uses GPS data to
follow the path the user takes throughout the duration of FMS data logging. Stop
detection utilizes jumps in the spatial and temporal data to obtain stops, and then
uses information from Wi-Fi, GSM, and the accelerometer to merge stops by de-
tecting periods where the user is still. Finally, it adds stops where mode detection
determines a change in mode of transportation.

3.2.2.2 Driving Style Detection - Overview

Driving style detection was conducted through two separate approaches. A more com-
plex, but potentially more extensible approach, involved the use of a signal processing
technique called Dynamic Time Warping (DTW). DTW is a classifier that is capa-
bale of ascertaining the similarity of two different signals, despite differences in speed
[23]. This technique allows us to record template signals representing sensor feedback
from aggressive driving in a test setting, and compare sensor data from a real trip to
the template signals. A second approach was through the use of initial orientation
calibration using accelerometers and hard thresholds on accelerometer magnitude.

3.2.2.3 Driving Style Detection - Dynamic Time Warping

The DTW method used in FMS Advisor relies on the passive mode implementation
within the paper by Johnson [23], analyzing data from sensors after the end of the trip.
While the paper relies on sensor fusion for more accurate results, the implementation
here relies mostly on acceleration data, since hard accelerations and decelerations are
the priority. The trip data is recorded at 25Hz, much higher than core FMS, to allow
for more accuracy in detecting events. The data is then filtered using a low-pass
Butterworth Filter [20] with a cutoff frequency of 1Hz, to remove potential noise
in the signal from bumps in the road or vehicle vibration. This cutoff frequency is
used by the paper [23], but the paper states that noise is highly dependent on road
conditions and that this was merely the best overall choice.

Event detection then takes place. This is done through the use of a simple moving
average (SMA) of the acceleration squared, as this allows for the magnitude of acceleration to be used. The moving average is computed over a window of size $k$, which was set to 15. This window size was chosen since it allows the average to be computed over less than a second (the frequency of logging is 25Hz). This is enough to account for noise without distorting the original signal. This window advances throughout the entire sensor data stream. If at a given time step the SMA is greater than an upper threshold, then an event is marked as started, and continues until the SMA goes below a lower threshold. The upper and lower threshold are determined empirically (see 4.2.4). The signal between the start and end times of the event are then compared to the template using DTW. The SMA itself is calculated using Equation 3.1.

$$SMA = \frac{g_x(i)^2 + g_x(i-1)^2 + \ldots + g_x(i-k-1)^2}{k}$$ \hspace{2cm} (3.1)

The classification is then performed. The DTW algorithm compares the two input signals: the template signal $X$ of size $m$ and the event found above $Y$ of size $n$. It then places both signals on an $xy$ grid of size $m \times n$. Each cell $(i,j)$ in the grid contains the absolute value of the difference between the values of the two signals at that point $\|X_i - Y_j\|$. DTW then attempts to find the shortest cost path from $(0,0)$ to $(m,n)$, with a cost $c$. If $c$ is below some event matching threshold (determined empirically), the event is determined a match.

The dynamic time warping method is more sophisticated in its determination of events than simple threshold methods. Unfortunately it has many empirically derived parameters, and the values of these parameters are difficult to determine across different vehicles and smartphone models (see 4.2.4). For this reason, a simpler threshold-based method was used for aggressive driving events.

### 3.2.2.4 Driving Style Detection - Threshold-based Methods

The threshold method [31] uses a more simple method that combines calibration of the initial orientation of the phone with hard magnitude thresholds on accelerometer
values. This allows for fewer empirically derived parameters to be required, and is therefore more likely to work across a range of vehicles and smartphones.

At the start of the trip, the initial orientation of the smartphone is recorded by the application. This is done by reading the accelerometer data for each axis, and reconstructing the orientation of the phone based on the expected gravitational acceleration. The roll and pitch of the phone are stored and are used to rotate incoming data during the trip on the fly to properly account for the starting orientation of the phone.

Once the data has been recorded for the whole trip, analysis proceeds along a sliding time window through the data, taking the simple moving average as described earlier with the empirically determined window of size $k$. Accelerometer thresholds are then used for each axis to determine whether or not an aggressive driving event has occurred, based on the SMA just calculated. These thresholds (Figure 3-10) are determined empirically, and are based on the values found in [31].

![Figure 3-10: Accelerometer thresholds, taken from [31].](image)

The particular driving events being detected are hard accelerations and decelerations, as these are related to poor energy efficiency. If an acceleration threshold is exceeded for an appropriate length of time $t$ (empirically determined by the paper to be 5 seconds), a hard acceleration is recorded and a notification is sent to the user of their inappropriate driving.

### 3.2.2.5 Data Logging and Collection

Logging of sensor data is an important part of FMS Advisor. This allows for further analysis of the trip after it has been completed, and for longer term tracking of different trends. The original FMS application focuses on accelerometer, GPS, and Wi-Fi data, and FMS Advisor expands this data collection to include the gyroscope.
and magnetometer sensors on the smartphone. These five sensors are collected by FMS Advisor through separate background loggers. In addition, the frequency of logging is increased dramatically for more fine-grained analysis. This higher frequency logging only takes place during an actual trip; after a trip has concluded, frequency returns the core FMS standards.

Logging takes place in temporary data structures until an explicit call to stop logging is sent from a different application component. When this happens, the sensor logs are written into JSON files stored on the smartphones internal storage. These JSON files are then compressed into zip files and split into evenly-sized chunks to be transmit to the backend server. This is done to prevent interrupted uploads of large files from internet disconnects, which would require re-sending the file. When all zip files are uploaded, the logger notifies the main application and clears the temporary files stored on the smartphone. These techniques utilize the core FMS application network functions, which have already been optimized for network usage.

### 3.2.2.6 Vehicle Occupancy Detection

Vehicle Occupancy Detection, another aspect of trip validation, is done using standard Bluetooth connections. A database of device names and MAC addresses associated with FMS Advisor users is stored on every device using the app, and a Bluetooth scan is conducted from the device. Based on the number of recognized devices observed in the scan, occupancy of the vehicle is determined. This will eventually tie in the token rewards in the marketplace because of the energy saved through ridesharing. There are also plans for the construction of a server-side database of MAC addresses registered with FMS Advisor, so the list of users could be updated automatically.

Using standard Bluetooth connections is unfortunately the only method of compatibility with older devices, particularly those old enough to be running Android 4.0. One potential issue is MAC address spoofing, where a malicious user modifies the MAC address broadcast by the device to be that of a registered FMS Advisor user. Though MAC address spoofing is possible, this solution is likely good enough for our purposes. Other options in the future once recent hardware advances become
more widespread include Bluetooth LE (low energy) and NFC (near-field communications). These would allow for scans to occur more periodically in the background, or for more painless, contact-based connections without a scan at all that would be harder to spoof. Unfortunately at this time it is not feasible to use these other options due to support for older hardware.

The limitations described above require occupancy detection to be run only at the start of a trip. Unfortunately, because of the processing power required by standard Bluetooth, very frequent scans cannot be conducted without strain on the application. In the future, more scans throughout the trip (i.e. every five minutes) are planned for more accurate detection. The data received from vehicle occupancy detection is not currently linked with user optimization, but is planned to be integrated in the future.

### 3.2.3 Marketplace

The Marketplace will interface with the Tripod projects Token Rewards Market module. It will allow users to spend tokens earned via their transportation decisions on various rewards. The marketplace will list the current tokens earned by the user, as well as all of the redeemable rewards given the current token total. The user will be able to sort the list of rewards by token value as well as by popularity of the reward from other users purchasing histories. The Marketplace interface has not yet been developed, but all the components under development are designed according to this requirement.

### 3.3 User Preferences Database

The core FMS application communicates with a backend data server which allows logged data from the application to be stored externally for further analysis by the research team. This backend data server is written in Rails [28], a Ruby-based application server that allows easy communication with databases for data storage. The data is logged in a MySQL [26] database with separate tables for each sensor logged.
on the smartphone app. In addition, login information (username and password), login times, and results to survey data are stored in separate tables.

FMS allows communication to the backend server through two primary methods: a web interface and a REST API. The web interface allows for interactive viewing of the locations visited by the user while the app is active, as well as full survey data related to the app. In addition, settings related to the user (such as password and logging preferences) can be modified through the use of the web interface.

The second method of communication is through a REST API exposed by Rails. This is the method the app uses to communicate with the backend, sending data in JSON format to the server. After receiving sensor data, the backend server processes the data, parsing the JSON into data appropriate for the MySQL tables. This is conducted in the background, allowing existing data to still be viewed through the web interface.

To accommodate FMS Advisors additional data storage requirements, a modified version of the FMS backend server database was deployed. This one was placed locally in Boston, as opposed to the core FMS server in Singapore. The online interface is slightly modified but is otherwise matches core FMS, while the REST API allows for additional sensor data to be stored (such as the gyroscope and magnetometer). The scripts are rewritten versions of core FMS scripts to allow processing this additional data, and to store data of differing granularity into the same tables.

3.4 System Optimization and Fallback Route Determination

The SO server is built to serve routes determined by the DynaMIT project [17] based on parameter submission from FMS Advisor. While this backend is still in construction, FMS Advisor provides a fallback mechanism using the Google Directions Javascript API [10] to determine multiple routes, using the parameters selected in the interface. These routes are requested through the Volley module, similar to com-
munication with the data backend, and are passed to the UO for ranking based on user preferences.

The routes returned through the Google Directions API allow for breakdown into individual waypoints along the route. These waypoints are necessary for later passing into Google Maps Navigation for the turn-by-turn navigation feature.

### 3.5 User Optimization

The User Optimization (UO) [30] is an algorithm that ranks a list of provided routes based on user preferences. To do this, it requires a list of possible routes from a given origin to a given destination that differ on certain key parameters. These parameters could include mode of transportation, the actual route taken, or the presence of tolls or other fees. The UO, on a per-user basis, keeps track of past routes taken and uses these to develop a unique user preferences matrix along these different parameters.

The user preferences matrix is then used to score the available routes for the trip based on the obtained parameters from the SO. A re-ranking by score is then sent back to FMS Advisor for actual display in the route list. The final route selected is then sent to the UO for an update of user preferences.

One major decision made was where the UO algorithm was to be run. There are advantages to running the algorithm directly on the phone, including a less complicated architecture and less data usage. However, battery life and computation time are a concern, depending largely on the size of the preference matrix and number of available routes. Profiling (see 4.2.6) was performed to make the decision. In the latest version of FMS Advisor, the UO algorithm runs directly on the phone. However, as the UO algorithm continues to get more sophisticated, there are plans to transition this code to a separate UO server.
Chapter 4

Evaluation

The primary method of evaluation for FMS Advisor and its related components involved testing for functionality and usability. Functionality was determined by whether the full workflow of the application could be accessed across a range of devices without crashes or incorrect behavior. Usability was determined through judgement of general ease of use, aesthetics, and adherence to the initial design mockups.

4.1 Methodology

Testing of FMS Advisor was important to quickly resolve issues, tune parameters, and make sure logged data met appropriate specifications. To do this, a combination of user testing in groups, surveys, and more controlled experiments were run. Different components of FMS Advisor utilized different evaluation methods reflecting subjectivity of evaluation and ease of repeating experiments.

Group testing involved setting up specialized sessions that encouraged immediate user feedback and quicker introductions to the use of the app. This was limited to a select few sessions due to time, but was successful in obtaining first impressions and identifying major problems quickly. Surveys served as a more long-term source of feedback. They were created through the use of Google Forms, which allowed for easily tabulation of survey results. The surveys were structured to allow for single-issue reporting, though more detailed issues required direct contact.
Bug tracking was managed through the use of JIRA, a product from Atlassian [13] that allows for fine-grained management of issues and distribution of responsibility. JIRA allows classification of issues, comment threads, and assignment of bugs to specific users. In addition, management features such as filtering, searching, and organization by tags make it easy to group together similar issues and to prevent issue duplicates.

4.2 Component Testing

This section will detail testing of the individual FMS Advisor components. The three major components are the user interface (UI), the backend server, and the trip validation algorithms. Another major test performed was that of user optimization performance, which was necessary in determining the location of the algorithm (either client or server-side).

4.2.1 Experimental Setup

The majority of the testing for FMS Advisor was done on a physical Samsung Galaxy S4 running Android 4.2.2 Jelly Bean. The application was developed in Java 8 using the OpenJDK version, and used Android API version 22. Android Studio 2.2 was used as the development environment, with the official Android emulators being used on occasion for sporadic testing. The choice of hardware platform was made due to available hardware for testing. Software choices utilized the latest version of development tools while remaining compatible with a wide range of potential user hardware.

4.2.2 UI Testing

The testing of the user interface was a key aspect of the overall evaluation of FMS Advisor, as explicit interaction from the user was an important goal in the project. The Trip Planner is the major UI component to FMS Advisor, so testing largely relied
on the workflow of this portion of the app. UI testing took the form of appraising the functionality of the UI and design issues related to usability and general consistency.

To do this, live user testing sessions were conducted to introduce new users to FMS Advisor and to walk them through the general workflow of the application. During these testing sessions, basic compatibility with users phones was evaluated, and any major issues (app crashes, functionality issues) that appeared during the session were noted. After going through general application usage and allowing users to explore the app, a survey (Figure 4-1) was sent to all testers to allow them to evaluate the application usage long-term.

![FMS Advisor Bug Report](image)

**Figure 4-1:** Google Survey used for bug reporting.

While the in-person session was helpful in diagnosing bugs, the survey itself didn’t see much direct use. In general, detailed bug reports were hard to fill out because many issues were crashes or provided little feedback to the user. There was also little incentive to testers to make active use of the application, so a more structured program of use is likely necessary in the future to help adoption in the testing phase. Because of these issues, bug reproduction and debugging had to be done manually.
to diagnose which specific components caused errors, and much of the testing came internally from within the FMS Advisor team.

In terms of actual breakdown of the issues reported, most of the issues were deemed bugs, with a few enhancements reported as well. Within the bugs, there was a further breakdown between critical (functionality-impeding) and more minor issues. The breakdown is reported below (Figure 4-2).

![Issue Breakdown](image)

Figure 4-2: Issue breakdown by type.

### 4.2.3 Backend Testing

Backend testing was essential to ensure that the data being logged by FMS Advisor during a trip was properly stored at essential times and that the granularity of the data being stored matched FMS Advisor standards. This was done through repeated testing of particular routes through the Trip Planner, and observation of the data store on the backend server.

The route was set in a way such that the geofence for the destination could be entered immediately. The route is pictured below (Figure 4-3). The actual choice of route was not important; the goal was whether data was logged correctly and sent...
at the end of the trip. The slight buffer period allowed for a small amount of sensor
data to be gathered, and sent over immediately with little delay due to the size. The
appropriate tables for the sensor data were then examined to see if the granularity
of data and the appropriate timestamps matched the FMS Advisor specifications.
Files are uploaded by the smartphone in batches to prevent upload errors (3.2.2.5); statistics on this processing (Table 4.1) were also found. The processing time refers
to the time the backend server takes to extract data from the JSON files and load
the data into its database.

<table>
<thead>
<tr>
<th>Average processing time</th>
<th>143.13 seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average number of processed files in one batch</td>
<td>37.82</td>
</tr>
</tbody>
</table>

Table 4.1: Backend server statistics.

Initially many server issues existed due to hardcoded values in the server codebase
regarding URLs, including redirecting to core FMS servers. User account signup also
had to be reconfigured to allow users to sign up again through the FMS Advisor web
interface, and the tables in the backend server reset to reflect those signups. After these issues were resolved, the “user_uploaded_files” table was used to see if sensor logs were being uploaded at the correct time, and that the background processing tasks ran to completion. The higher frequency of the data ended up posing no problems for the database, which allowed the table to contain timestamps of different frequencies.

### 4.2.4 DTW Driving Style Detection Testing

Both driving style detection methods outlined in the design section (see 3.2.2.3) required empirically derived parameters to determine thresholds, and so testing the algorithms under specific situations was necessary. Dynamic time warping also requires template signals for comparison with the sensor data recorded during a trip, so additional data needed to be logged.

For obtaining the template signals, data was recorded from three sensors: the accelerometer, the gyroscope, and the magnetometer. While recording, a vehicle was driven to an empty parking lot and performed aggressive driving maneuvers while the app was running. Sharp right turns, left turns, u-turns, and aggressive accelerations and decelerations were performed and the signals on each sensor recorded. All of these actions were performed with the phone placed parallel to the ground, facing the front of the vehicle. This was done to orient the initial axes correctly, since x, y, and z data are all used in driving style detection.

To actually test DTW driving style detection after implementation, and to help tune parameters, FMS Advisor was run on a trip on MIT’s shuttle system [15]. This trip went on a loop from MIT campus to various locations in Cambridge, MA. The focus of the trip was on verifying that signal data from the smartphone was acceptable to compare to template data using the DTW algorithm. Hard accelerations and decelerations were prioritized, and the accelerometer was logged during the trip. The smartphone was mounted on a seat, in a flat position, to remove any orientation changes during the trip. The accelerometer sensor was measured (Figure 4-4) during the trip to help determine parameters such as upper and lower threshold for events.

Unfortunately, the data was subject to noise during the trip, largely due to the
high number of bumps and internal vibration of the shuttle. To help mitigate noise, a low pass filter was applied on the sensor data (Figure 4-5) to help extract the relevant signal.

Even with the lowpass filter, the data remained too noisy. With excessive filtering,
noise would disappear but would obscure the original signal, losing too much signal data in the process. Resolving this problem would be difficult in larger vehicles such as buses and shuttles, but a stable mounting surface within smaller vehicles like cars would likely reduce noise. This made it difficult to compare to template signals, as DTW required fairly accurate sensor data for the algorithm to operate. While short-term sensor logging (as was used in the template generation) was fairly representative, long-term logging had significant noise issues. For this reason, a threshold-based method was explored for driving style detection.

4.2.5 Threshold-based Driving Style Detection Testing

Threshold-based driving was tested by comparison with data received from on-board devices (OBDs). The particular model used was the Automatic Pro [7], which provides access to its data through the use of a streaming, real-time API. These OBDs can be placed in vehicles and are able to measure ignition events while driving. In particular, the OBDs can also distinguish hard acceleration events, dictated by the device-specific threshold of 11km/h, or 0.3 times the force of gravity.

The streaming events were queried using a script and output to a local JSON file, which was then processed through another sequence of scripts. These allowed hard acceleration events to be filtered out, and the location and magnitude of these events to be noted. Over the course of a week-long experiment, the events in (see Table 4.2) were noted, with (see Figure 4-6) marking the locations of these events.

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Latitude/Longitude</th>
<th>Acceleration (g)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-11-15 14:57:31</td>
<td>42.319173, -71.204337</td>
<td>0.422</td>
</tr>
<tr>
<td>2016-11-15 15:07:31</td>
<td>42.326553, -71.141183</td>
<td>0.309</td>
</tr>
<tr>
<td>2016-11-16 22:18:58</td>
<td>42.332894, -71.124725</td>
<td>0.309</td>
</tr>
<tr>
<td>2016-11-17 22:39:56</td>
<td>42.319452, -71.179785</td>
<td>0.308</td>
</tr>
<tr>
<td>2016-11-18 14:41:06</td>
<td>42.329987, -71.126549</td>
<td>0.337</td>
</tr>
<tr>
<td>2016-11-19 04:59:25</td>
<td>42.26882, -71.219988</td>
<td>0.309</td>
</tr>
<tr>
<td>2016-11-19 16:41:21</td>
<td>42.319098, -71.204349</td>
<td>0.393</td>
</tr>
<tr>
<td>2016-11-21 17:46:38</td>
<td>42.319156, -71.204291</td>
<td>0.365</td>
</tr>
<tr>
<td>2016-11-22 00:00:32</td>
<td>42.353725, -71.105055</td>
<td>0.309</td>
</tr>
</tbody>
</table>

Table 4.2: Hard acceleration data from OBDs.
Comparison of this data was done with users running core FMS on their smartphones. Unfortunately, the users only had iOS devices, and so FMS Advisor (with its higher frequency data logging) could not be used. The accelerometer data gathered by FMS was then analyzed for the test period, in an attempt to correlate accelerometer readings with ignition readings from the OBDs. Figure 4-7 shows the accelerometer readings with vertical lines superimposed representing hard acceleration events detected by the OBDs.

Analysis was done on the average accelerometer readings after these events happened for a timescale of 3 seconds. The readings (see Table 4.3) were offset by a magnitude of 1 for gravitational force. While some of the data reflected a threshold between 0.2 and 0.3g, other results were not as large. This likely is due to the low frequency of data collected, which made the variance of the results high. Given the difficulty in establishing thresholds through this method, testing DTW-based driving style detection would also be infeasible due to the low frequency of the signal data.

Through the use of FMS Advisor in a future experiment, with higher frequency data collection, an acceleration threshold could be determined for threshold-based detection. This higher frequency would also allow for more detailed signals that
could be used to polish DTW-based driving style detection as well. Further testing also needs to be done to confirm these thresholds work in actual vehicles, but the analysis done so far should help inform parameters to a reasonable degree.

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Acceleration (g)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-11-16 22:18:58</td>
<td>1.0301897</td>
</tr>
<tr>
<td>2016-11-17 22:39:56</td>
<td>1.021827</td>
</tr>
<tr>
<td>2016-11-18 14:41:06</td>
<td>1.00747</td>
</tr>
<tr>
<td>2016-11-19 04:59:25</td>
<td>1.0237257</td>
</tr>
<tr>
<td>2016-11-19 16:41:21</td>
<td>1.0021825</td>
</tr>
<tr>
<td>2016-11-21 17:46:38</td>
<td>1.052587</td>
</tr>
</tbody>
</table>

Table 4.3: Hard acceleration data from FMS, aligned with OBD data.

### 4.2.6 User Optimization Performance

To test the impact of the UO algorithm on performance, Traceview was used. Traceview [29] is a built-in profiling tool to the Android Studio development environment, and allows individual method calls to be analyzed for time taken, as well as the overall percentage of CPU power used relative to the rest of the application. This helped determine the minimal impact of the preliminary UO algorithm, as areas such as the
UI and data communication played a much larger role in CPU usage. Given the minimal CPU usage, it was determined that battery life was also marginally impacted, leading to the decision to keep the UO on the phone at its current state.

Figure 4-8: Traceview profiling showing UI calls taking the majority of CPU time.

4.2.7 Future Testing Strategy

As described earlier, much of the evaluation is still in preliminary stages and requires more rigorous testing of both the user interface and the validation algorithms. This section will describe some potential future strategies for delivering better and more comprehensive results.

For user interface testing (see 4.2.2), work needs to be done to allow for more widespread adoption of FMS Advisor in day-to-day use. This would allow issues to better surface, as more time spent with the application will put it through a wider variety of usage scenarios. Currently, while group sessions work in highlighting easily discernible issues with the application, actual use of the application on trips is needed to verify all components of the application work across a range of devices. The survey
method described earlier was not successful on its own, and so I propose incentivizing
the UI testing. By providing rewards to testers who report issues (a “bug bounty
program”), survey results would likely improve.

Both driving-style detection methods still have pieces that need to be verified,
and parameters that need to be defined and evaluated. Dynamic time warping (see
4.2.4) needs further testing to ensure the templates are accurate, that the parameters
derived in the paper [23] are appropriate, and that the algorithm works correctly to
match the recorded templates. The threshold-based detection (see 4.2.5) needs a more
controlled experiment to determine whether the accelerometer thresholds determined
in the paper [31] are appropriate within FMS Advisor. Both methods had issues
regarding the granularity of the data gathered and the difficulty in reducing noise
from the sensor data. Another issue was having easy access to a car for testing, and
being able to use FMS Advisor directly in conjunction with on-board devices.

Noise can be reduced through the use of phone mounts, and mandating that
phones are kept in a fixed orientation throughout the trip. This reduces the impact
of vibrations from the vehicle, and makes sure readings are not skewed by orientation.
Large vehicle testing was the primary method used so far, and was found to produce
too much noise to be accurate. An optimal controlled experiment would place the
smartphone running FMS Advisor (with high frequency logging) mounted in a car,
with an on-board device to help verify results. Ideally, this car would be accessible
on multiple trips, to allow for quicker feedback in updating the application.
Chapter 5

Conclusion

The goal of the Tripod project is to fully leverage smartphone technology to provide a system to encourage the use of energy efficient transportation. My contribution, FMS Advisor, provides a smartphone user an interface to the determined incentives, through a trip planner interface. It also provides background trip validation and data logging that can be used for further analysis. From the initial proposal, many aspects of FMS Advisor were completed and the current version of the application provides an end-to-end experience that allows a user to plan a trip, obtain turn-by-turn navigation, and have sensor data logged with trip validation efforts in the background.

The trip planner interface is largely feature-complete, with the user interface largely matching the original design proposed for FMS Advisor. Parameter input, route list selection, and launch of turn-by-turn navigation are all functional, and the general aesthetics follow the original design. A few minor features, such as route display on a map before final selection, remain unimplemented but are planned for the near future. Rigorous evaluation of the interface is still in preliminary stages, with many superficial issues ironed out. However, compatibility across a wide range of devices has not been fully tested yet, and is a priority for future updates.

Certain aspects of trip validation are complete, while other parts are still in the testing phase and require further refinement before they can be properly deployed. Core FMS features leverage the existing codebase. Vehicle occupancy detection was
completed and is functional, with further improvements involving data storage server-side planned. Driving style detection faced roadblocks during its creation, due largely to the complexity of the algorithms involved, inaccuracy of sensor data, and difficulty in testing out implementations. Evaluation of driving style detection was unsuccessful, with each experiment being susceptible to noise in different ways. While implementation of the algorithms is complete, free parameters need to be empirically verified to match expected results. A better controlled experiment is described in detail in 4.2.7.

The scope of the FMS Advisor project expanded during the course of development, adding many additional features. User optimization integration became a key component, as the algorithms were run directly on the smartphone device. Backend server modifications were also added to the FMS Advisor project, introducing a more complicated architecture to be handled by the team. Much of the time spent on FMS Advisor so far has been on the development of features, with testing still in progress. Future efforts in FMS Advisor will refine the existing features and provide a fully polished product ready for use on a large scale within the larger Tripod project.
Bibliography


