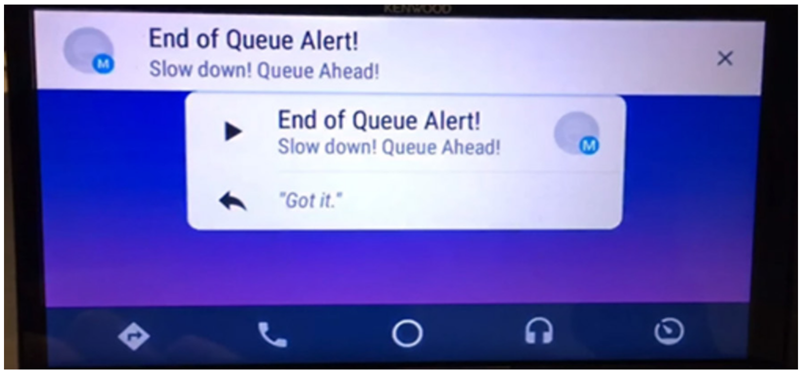
JOINT TRANSPORTATION
RESEARCH PROGRAM

INDIANA DEPARTMENT OF TRANSPORTATION
AND PURDUE UNIVERSITY

Back of Queue Warning and Critical Information Delivery to Motorists


End of Queue Alert!
Slow down! Queue Ahead!

End of Queue Alert!
Slow down! Queue Ahead!

"Got it."

Lingxi Li, Yaobin Chen, Renran Tian, Feng Li

SPR-4306 • Report Number: FHWA/IN/JTRP-2019/24 • DOI: 10.5703/1288284317102

RECOMMENDED CITATION

Li, L., Chen, Y., Tian, R., & Li, F. (2019). *Back of queue warning and critical information delivery to motorists* (Joint Transportation Research Program Publication No. FHWA/IN/JTRP-2019/24). West Lafayette, IN: Purdue University. https://doi.org/10.5703/1288284317102

AUTHORS

Lingxi Li, PhD
Associate Professor of Electrical and Computer Engineering
School of Engineering and Technology
Indiana University–Purdue University Indianapolis

Yaobin Chen, PhD
Director of TASI and Chancellor's Professor of Electrical and Computer Engineering
School of Engineering and Technology
Indiana University–Purdue University Indianapolis
(317) 274-4032
ychen@iupui.edu
Corresponding Author

Renran Tian, PhD
Assistant Professor of Computer Information and Graphics Technology
School of Engineering and Technology
Indiana University–Purdue University Indianapolis

Feng Li, PhD
Associate Professor of Computer and Information Science
School of Engineering and Technology
Indiana University–Purdue University Indianapolis

JOINT TRANSPORTATION RESEARCH PROGRAM

NOTICE

TECHNICAL REPORT DOCUMENTATION PAGE

1. Report No. FHWA/IN/JTRP-2019/24	**2. Government Accession No.**	**3. Recipient's Catalog No.**
4. Title and Subtitle Back of Queue Warning and Critical Information Delivery to Motorists		**5. Report Date** August, 2019
		6. Performing Organization Code
7. Author(s) Lingxi Li, Yaobin Chen, Renran Tian, and Feng Li		**8. Performing Organization Report No.** FHWA/IN/JTRP-2019/24
9. Performing Organization Name and Address Joint Transportation Research Program Hall for Discovery and Learning Research (DLR), Suite 204 207 S. Martin Jischke Drive West Lafayette, IN 47907		**10. Work Unit No.**
		11. Contract or Grant No. SPR-4306
12. Sponsoring Agency Name and Address Indiana Department of Transportation (SPR) State Office Building 100 North Senate Avenue Indianapolis, IN 46204		**13. Type of Report and Period Covered** Final Report
		14. Sponsoring Agency Code
15. Supplementary Notes Conducted in cooperation with the U.S. Department of Transportation, Federal Highway Administration.		

16. Abstract

Back-of-queue crashes are one of the main sources for fatal accidents on U.S. highways. A variety of factors including low visibility, slippery road surface, and driver distraction/drowsiness during highway cruising, all contribute to this type of fatal crashes. Thus, it is very important to improve the driver's situational awareness before they approach traffic queues on highways.

In this project, we develop a prototype in-vehicle back-of-queue alerting system that is based on the probe vehicle data from INDOT. Speed changes among different road segments are used to identify slow traffic queues, which are compared with vehicle locations and moving directions to detect potential back-of-queue crashes. This prototype system is designed to issue alerting messages to drivers approaching the highway traffic queues via an Android-based smartphone app and an Android Auto device. The performance of this system has been evaluated using the driving simulator and a limited number of on-road test runs. The results showed the effectiveness and benefits of this prototype system.

17. Key Words back-of-queue crashes, smartphone app, android auto, driving simulator study		**18. Distribution Statement** No restrictions. This document is available through the National Technical Information Service, Springfield, VA 22161.	
19. Security Classif. (of this report) Unclassified	**20. Security Classif. (of this page)** Unclassified	**21. No. of Pages** 25	**22. Price**

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

EXECUTIVE SUMMARY

Introduction

Recent INDOT JTRP studies have found that back-of-queue or end-of-queue crashes are one of the main sources for fatal accidents on the highway. A variety of factors, including low visibility, slippery road surface, and driver distraction/drowsiness during highway cruising, contribute to this type of fatal crash.

Based on probe vehicle data acquired through freight, smartphones, and in-vehicle GPS units, INDOT is able to monitor highway congestion, queue-starting locations, and queue lengths covering all interstate highways in Indiana. Although this information is available to the traffic managers, there are no methods that are currently available to effectively distribute information about potentially hazardous conditions to drivers on the road.

Based on the existing INDOT real-time queue-monitoring system, this project focused on developing a prototype end-of-queue alerting system for drivers approaching traffic queues on Indiana highways. More specifically, this research aimed to do the following:

- Investigate feasible solutions to implement end-of-queue alarms, considering both in-car and roadside options, and tune the alarm parameters through driver simulator-based studies.
- Determine the best practice for end-of-queue alarms by examining the technical requirements and limitations of different feasible solutions, with a focus on queue data acquisition, vehicle localization, communications, and adaptive in-vehicle alarms.
- Develop a prototype end-of-queue alerting system based on probe vehicle data and evaluate the benefits via a driving simulator–based study and a limited number of on-road driving tests.

Findings

- The establishment of INDOT's web service was very important for the IUPUI team, allowing them to fetch traffic data in real-time and develop algorithms for issuing the end-of-queue alerts to drivers on the road.
- To find efficient and effective ways to distribute hazardous road information to on-road drivers, multiple potential solutions were investigated, such as portable roadside message boards, in-vehicle smartphone apps, and vehicle-to-vehicle communications based on in-vehicle devices. After a thorough literature review and an evaluation of existing data sources, the chosen end-of-queue alerting mechanism was in-vehicle smartphone app-based alerts. The alert messages will be sent through Android-based smartphone apps via the 4G network and the Android Auto device.
- The driving simulator is a very useful tool for evaluating the benefits of the developed prototype end-of-queue alerting system. Subjects were recruited to be tested in traffic scenarios with and without alerts and with different driving statuses (normal, distracted, and drowsy). Driver and driving data were collected and analyzed.
- Initial investigation proves that this alerting system can reduce intensive driving behavior overall and has the potential to increase driving safety.

Implementation

To implement the end-of-queue alerting system on the Android-based smartphone, a Java-based application was developed. Google map API was used to show the map information on the smartphone. To monitor the delta-speed changes, web service from INDOT was used. An algorithm was developed to find the speed events based on the current GPS coordinates of the smartphone. To debug the application, a separate mechanism was developed to simulate the real-time data in computer programs. This application was also tested on different highways and associated log files were created for debugging purposes. The notification classes were used to show the alerts on the smartphone, including sound, text, and vibration features.

The Android devices were purchased and their functions were evaluated. The Android Auto device we used was the Kenwood DDX9704S display and the Android phone was the Samsung S7; both were powerful enough for our research needs. Based on the evaluation, two different methods for sending out notifications were tested. The first one was to send notification as a text message. Taking advantage of the powerful Android Auto app and Google Assistant, the notification can be clearly displayed and interactive. The second method was to mirror the screen of the phone to the Android Auto display. In this way, more intuitive information can be shown. Both methods have pros and cons and are suitable for different needs. The smartphone app and Android Auto device were road tested. Both devices functioned well on the highway during the test runs.

The driving simulator in the TASI laboratory at IUPUI was set up for the testing of the prototype system. A platform was designed to utilize the driving simulator system (HyperDrive) and provide the mobile application with a simulated real-world view of Indiana highway traffic. Five subjects were asked to participate in tests designed to evaluate the effects of having an end-of-queue alerting system in the driving simulator. Each subject was asked to perform six driving tests in total. Subjects drove in the driving simulator under normal, distracted, and drowsy conditions with and without alerts. The driving simulator was able to collect braking and crashing data when the subject approached the queue. The performance of the prototype system was evaluated through the comparison of the data from the two test categories.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Background

Recent INDOT JTRP studies (Mekker et al., 2020) concluded that highway congestion is causing a much higher crash rate compared to uncongested driving conditions. For the highway congestion, back-of-queue or end-of-queue crashes are one of the main sources for fatal accidents and account for about 13% of all fatal accidents (Mekker et al., 2020). A variety of factors including low visibility, slippery road surface, and driver distraction/drowsiness during highway cruising, all contribute to this type of fatal crashes.

The role of common ADAS (advanced driver assistance systems) features such as adaptive cruise control (ACC) and lane-keeping assist (LKA) is two-sided in these back-of-queue crash scenarios. On one hand, some on-board sensors may help issue imminent crash warnings or braking assist to mitigate the crash consequences. On the other hand, these features may also cause increased driver distraction and inattention that potentially delay appropriate driver responses. Researchers have developed and evaluated end-of-queue alerting systems (Tampère et al., 2009; Nowakowski et al., 2012) showing good potential in smoothing the transition from free flow to congestion, as well as improving drivers' situational awareness. However, the published studies lack a systematic investigation on alarm timing, modality, frequency; driver status; and the corresponding driver responses, which significantly affect the effectiveness of the crash warnings (Lee et al., 2002).

Based on probe vehicle data acquired through freight, smartphones, and in-vehicle GPS units, INDOT is able to monitor highway congestion, queue starting locations, and queue lengths (Mekker et al., 2017) covering all interstate highways in Indiana, as illustrated in Figure 1.1. Although this information is available to the traffic managers now, there are no currently available methods to effectively distribute these potential hazards to on-road drivers.

1.2 Research Objectives and Approaches

Based on the existing INDOT real-time queue-monitoring system, this project focuses on developing a prototype end-of-queue alerting system for drivers approaching congestion queues on Indiana highways. More specifically, this proposed research aims to do the following:

- Investigate feasible solutions to issue end-of-queue alerts, considering both in-car and roadside options, and to tune the parameters through driver simulator-based studies.
- Determine the best practice for end-of-queue alerts by examining the technical requirements and limitations for different feasible solutions, and focusing on the aspects of queue data acquisition, vehicle localization, communications, and adaptive in-vehicle alarms.
- Develop a prototype end-of-queue alerting system based on the probe vehicle data and evaluate the benefits via a driving simulator-based study and on-road test runs.

1.3 Research Team

Transportation Active Safety Institute (TASI), Indiana University-Purdue University Indianapolis (IUPUI), 723 W. Michigan Street SL-160, Indianapolis, IN 46202, USA.

Principal/Co-Principal Investigators

- Dr. Yaobin Chen (PI), Professor of Electrical and Computer Engineering, responsible for overall project management and operation.
- Dr. Lingxi Li, Associate Professor of Electrical and Computer Engineering, responsible for overall project technical operation.
- Dr. Feng Li, Associate Professor of Computer Information and Graphics Technology, responsible for probe vehicle data acquisition and communications.

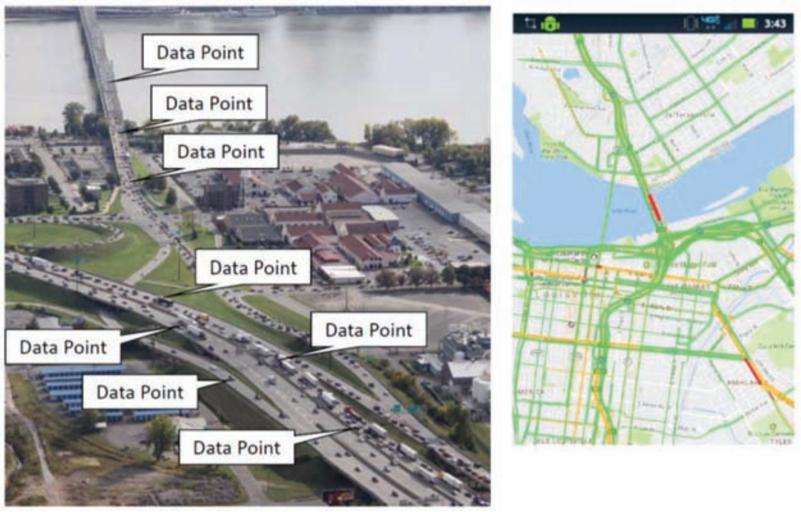


Figure 1.1 INDOT real-time queue monitoring system based on probe-vehicle data.

- Dr. Renran Tian, Assistant Professor of Computer Information and Graphics Technology, responsible for driving simulator-based study, data collection, and data analysis.

Postdoc Researcher

- Dr. Jue Zhou, responsible for supporting all project tasks.

Student Research Assistants

- Keyu Ruan, PhD student, Electrical and Computer Engineering, IUPUI.
- Dan Shen, PhD student, Electrical and Computer Engineering, IUPUI.
- Zahra Yarmand, M.S. student, Computer Information and Graphics Technology, IUPUI.
- Tina Chen, M.S. student, Electrical and Computer Engineering, IUPUI.

Student Hourly Workers

- Hamidreza Lotfalizadeh, PhD student, Electrical and Computer Engineering, IUPUI.
- Harshitha Veeramachaneni, M.S. student, Electrical and Computer Engineering, IUPUI.
- Alex Bemis, B.S. student, Electrical and Computer Engineering, IUPUI.

1.4 Organization of the Report

This report documents the research objectives, a detailed description of research methods, and the results for the project year. Each section in the report corresponds to one specific task.

2. PROBE VEHICLE QUEUE DATA ACQUISITION (TASK 1)

2.1 Introduction

The real-time traffic data of the highway delta-speed events on Indiana interstates is prepared by the INDOT team. This data is provided in a JSON format and gets updated every minute. This data includes different parameters of the event such as coordinate, direction, speed, name of the interstate, etc. We developed computer programs in Java to call the web service and fetch the data every minute. In our program, we handled repetitive event points. In particular, we followed a two-phase project plan: (1) Testing the core algorithm using manually generated GPS coordinates and demo map data in the simulated environment (shown in Figure 2.1). (2) Issuing alert messages to drivers by fetching the real-time traffic data from INDOT web service (shown in Figure 2.2).

2.2 Technical Approach

To access the data prepared by INDOT in our Java application, we created several classes related to the outputs of the web service. These classes were used to contain the information we receive from the JSON file and have the same properties. In particular, we developed several Java functions to convert JSON data files into Java classes. After that, we used a Java thread to send a request to the web service to fetch the data. This thread is responsible for reading the data from the web service and filling out the classes. After this step, we are able to access the latest information about the delta speed events in our Java program. This data will then be used in speed event detection algorithms. For the simulated environment, we fill the same classes with the data we get from the JSON file that contains fake delta speed events.

2.3 Summary

With the help of INDOT web service, we have access to the real-time data of all the delta-speed events on Indiana interstate highways in our application with a refresh rate of one minute. We developed computer programs in Java to fetch the data in real-time. This data is one of the most important inputs for the algorithms we developed to issue the alerts for this project.

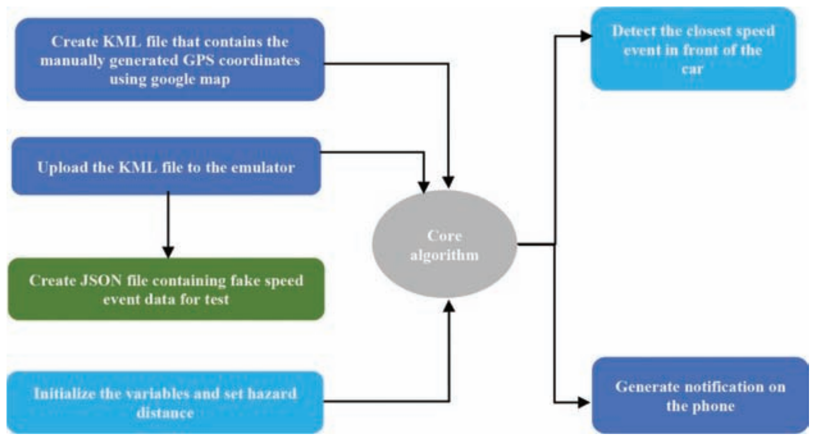

Create KML file that contains the manually generated GPS coordinates using google map
Upload the KML file to the emulator
Create JSON file containing fake speed event data for test
Initialize the variables and set hazard distance
Core algorithm
Detect the closest speed event in front of the car
Generate notification on the phone

Figure 2.1 Project plan—phase 1 in simulated environment.

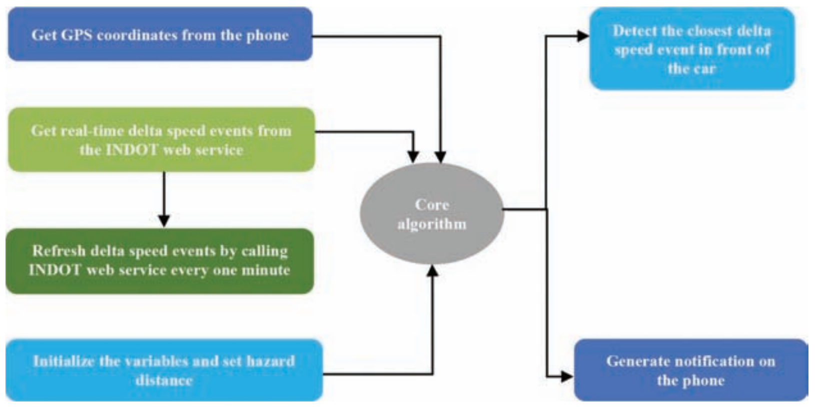

Get GPS coordinates from the phone
Get real-time delta speed events from the INDOT web service
Refresh delta speed events by calling INDOT web service every one minute
Initialize the variables and set hazard distance
Core algorithm
Detect the closest delta speed event in front of the car
Generate notification on the phone

Figure 2.2 Project plan—phase 2 in real traffic environment.

3. INVESTIGATION OF FEASIBLE END-OF-QUEUE ALERTING SOLUTIONS (TASK 2)

3.1 Introduction

The objective of this task is to investigate feasible end-of-queue alerting solutions by considering typical highway traffic-queue scenarios, road characteristics, and environmental factors.

3.2 Technical Approach

Through a comprehensive literature review and discussions with the INDOT team, the main technical approach for this task is to deliver in-vehicle alert messages to drivers through smartphone apps and a 4G network. In particular, the Android Auto device is able to be connected to Android-based smartphones and is available in many passenger vehicles. Alert messages can be delivered efficiently to drivers approaching the queue on highway via smartphone app. Smartphone GPS data can be used to obtain vehicle locations and directions. API for these apps was investigated along with proper alerting mechanisms to drivers. The smartphones of drivers approaching the highway traffic-queue can receive alerting information so that the drivers can be aware of the upcoming hazardous traffic information.

3.3 Summary

In this task, we investigated a variety of feasible end-of-queue alerting solutions. For this project, the technical approach to deliver in-vehicle alert messages to drivers was through Android-based smartphone apps and the Android Auto device.

4. HARDWARE/SOFTWARE IMPLEMENTATION FOR PILOT ALERTING SYSTEMS (TASK 3)

4.1 Introduction

In this task, we aimed at developing the hardware/software required for the prototype end-of-queue alerting system. Related hardware/software was implemented for further testing and evaluation, which consists of the following:

- Hardware: We simulated a hardware environment by integrating an aftermarket multimedia receiver with the Android Auto device and an Android-based smartphone (Samsung Galaxy S7).
- Software: The main functions of the Android-based app application were the following:

  1. Accessing the real-time traffic data of the speed events from INDOT web service.
  2. Accessing the current interstate name using Google API.
  3. Obtaining vehicle locations and directions through GPS coordinates of the smartphone.
  4. Determining potential risks when the vehicle approaches highway queues.
  5. Issuing end-of-queue alerts to drivers via developed smartphone app.

4.2 Technical Approach

4.2.1 Software—Android Studio

Android Studio. We used Android Studio to implement the Android application. Android Studio is the official integrated development environment (IDE) for Android application development. It is based on the IntelliJ IDEA, a Java integrated development environment for software, and it incorporates its code editing and developer tools. Android Studio is available for Mac, Windows, and Linux desktop platforms. Since we used Macbook for coding in this project, we used the Mac version of the Android Studio.

Algorithm. In this section, we describe the algorithm that is used for the end-of-queue alerting system. In general, it works based on matching the current road name that the vehicle is on and its driving directions with the road name and road direction returned from INDOT web service. Using this matching, we can find delta-speed events on the current interstate and issue the alerts by measuring the distance of the vehicle to every delta-speed point. Some variables, including hazard distance, mode of execution, zoom rate, etc., are defined at the very beginning in the program and can be changed based on different needs. The following sections will give more details about each step. The flow chart of the algorithm is shown in Figure 4.1.

a. Accessing the latest delta-speed events. The first step of the algorithm is to fetch the real-time traffic data from INDOT web service. The output of this web service is a JSON file. For this purpose, we developed a Java thread to send a get request to INDOT web service to fetch the real-time delta-speed events data and fill out the Java classes. The data returned from INDOT web service have different properties. The most important properties that we used in the algorithm are summarized below. Figure 4.2 shows a sample JSON file.

- Coordinates: This is a set of latitude and longitude that specify the location of that event.
- Direction: The direction of the interstate where the event occurs.
- Road: The name of Interstate where the event occurs.

b. Using GPS coordinates to find the location. We used the GPS coordinate captured from the smartphone to determine the location of the vehicle. Clearly, the GPS coordinate changes frequently when the vehicle is moving. If we need to test with the emulator, we have to upload the KML file containing the GPS

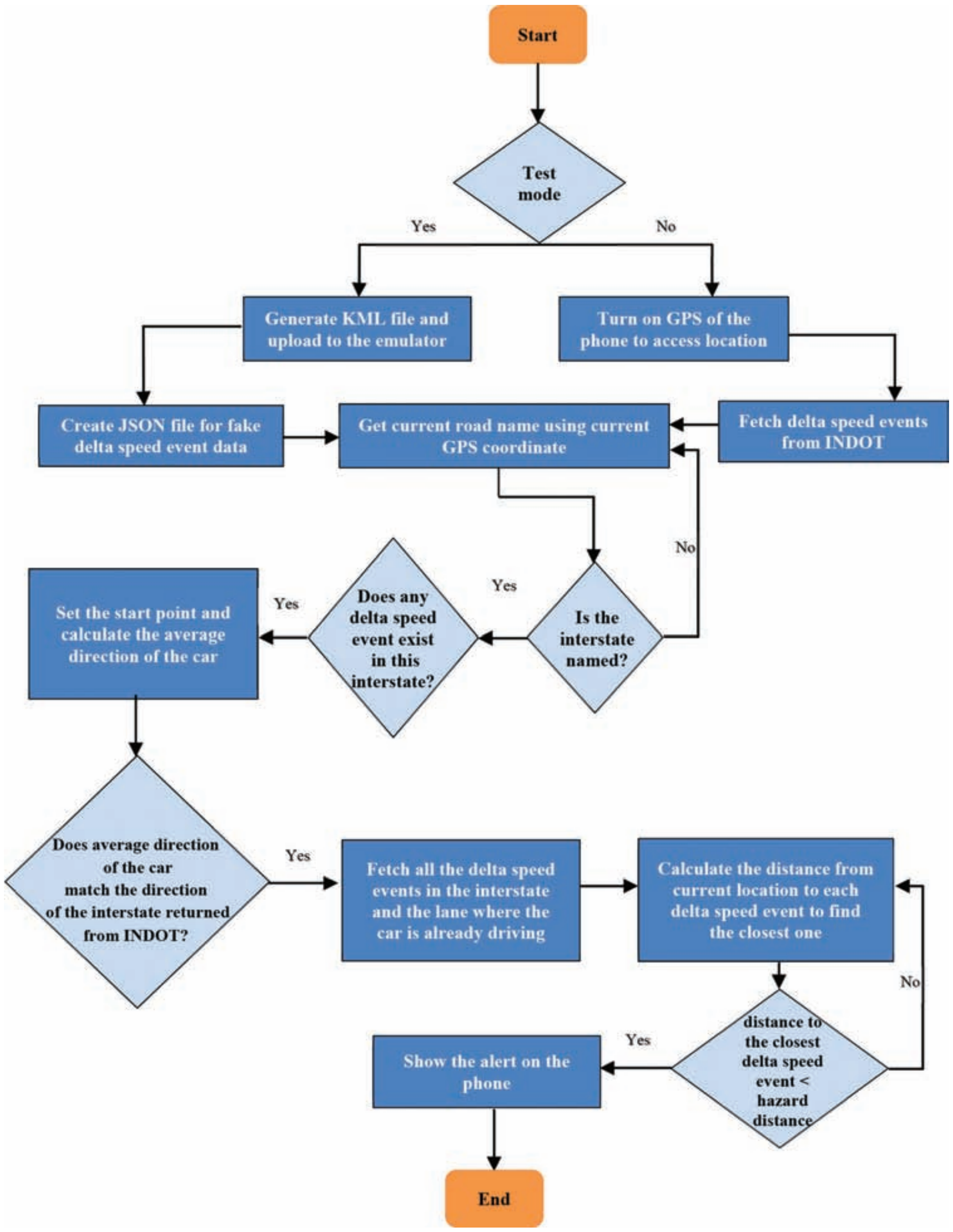
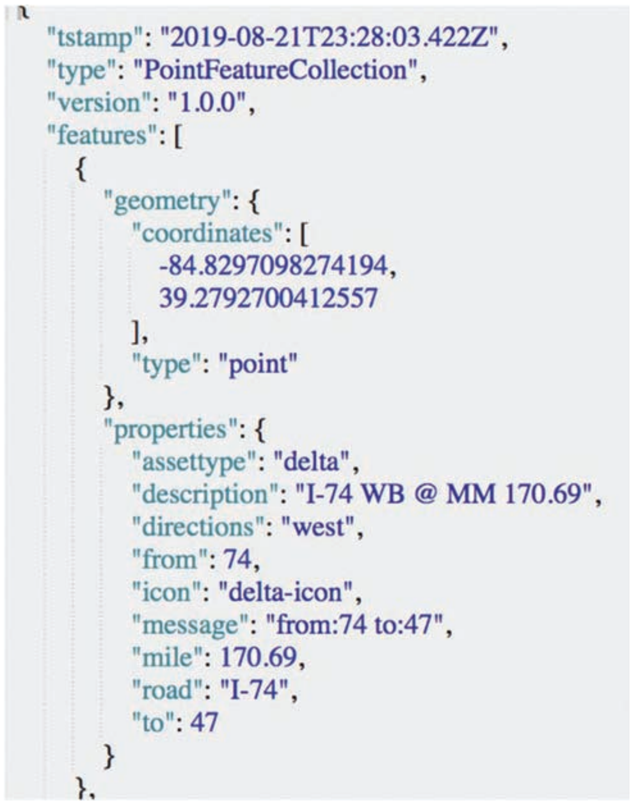

Start
Test mode
Yes
No
Generate KML file and upload to the emulator
Turn on GPS of the phone to access location
Create JSON file for fake delta speed event data
Get current road name using current GPS coordinate
Fetch delta speed events from INDOT
No
Yes
Yes
Set the start point and calculate the average direction of the car
Does any delta speed event exist in this interstate?
Is the interstate named?
Does average direction of the car match the direction of the interstate returned from INDOT?
Yes
Fetch all the delta speed events in the interstate and the lane where the car is already driving
Calculate the distance from current location to each delta speed event to find the closest one
No
Yes
Show the alert on the phone
distance to the closest delta speed event < hazard distance
End

Figure 4.1 Flow chart of the algorithm.



```
  "tstamp": "2019-08-21T23:28:03.422Z",
  "type": "PointFeatureCollection",
  "version": "1.0.0",
  "features": [
    {
      "geometry": {
        "coordinates": [
          -84.8297098274194,
          39.2792700412557
        ],
        "type": "point"
      },
      "properties": {
        "assettype": "delta",
        "description": "I-74 WB @ MM 170.69",
        "directions": "west",
        "from": 74,
        "icon": "delta-icon",
        "message": "from:74 to:47",
        "mile": 170.69,
        "road": "I-74",
        "to": 47
      }
    },
```

Figure 4.2 A sample JSON file.

coordinates of the simulated phone. Based on the data, we developed a mechanism to find delta-speed events on the driving direction of the vehicle.

c. Finding the road name. To find the current road name where the vehicle is on, we used the Google API. This API gets the current GPS coordinate as an input and returns a list of addresses related to this coordinate. We created a function in the program to detect if the road name returned from the API is the name of interstate or not. If the returned name is an interstate, we find its name; if not, we ignore the GPS coordinate.

d. Matching road names to find delta-speed events. When we get the interstate name from the API, we matched this name with the data returned from INDOT web service to determine if there is any speed event on the interstate. If there is an event, we set the first point as a start point and use it to calculate the direction of the vehicle in the next step.

e. Determining vehicle direction. We used a series of GPS coordinates of the vehicle to determine the driving direction of the vehicle on interstate highways. In parallel, a thread is developed to check whether the driving direction of the current interstate matches with the direction that the web service returns. If we find a match, then we are able to fetch the delta-speed events that are currently detected.

f. Calculating distance. After we fetch all delta-speed events, we have to detect the nearest event on the driving direction of the vehicle to issue the alert. For this purpose, we created a function inside the Java program to calculate the distance from the current location to every delta-speed event that we are detecting. Then we are able to find the nearest delta-speed event to issue the alert.

g. Issuing alerts. Hazard distance is a distance we select to issue the alert, which is a parameter we can adjust. We used the notifications on the smartphone to show the alerts, which has different properties such as sound, vibration, and text. An exemplary text notification is shown in Figure 4.3.

Testing and debugging. There are two ways of testing and debugging the application.

a. Using the Samsung Galaxy S7 smartphone. We can test and debug the program using the smartphone. Many functionalities such as generating alerts can be tested without the need to move. However, we need to drive on the highway to test the algorithm and the app.

b. Using emulator. One of the most important tools of the Android Studio is the emulator. An emulator is like a virtual Android device. It is a hardware device or a program that pretends to be another

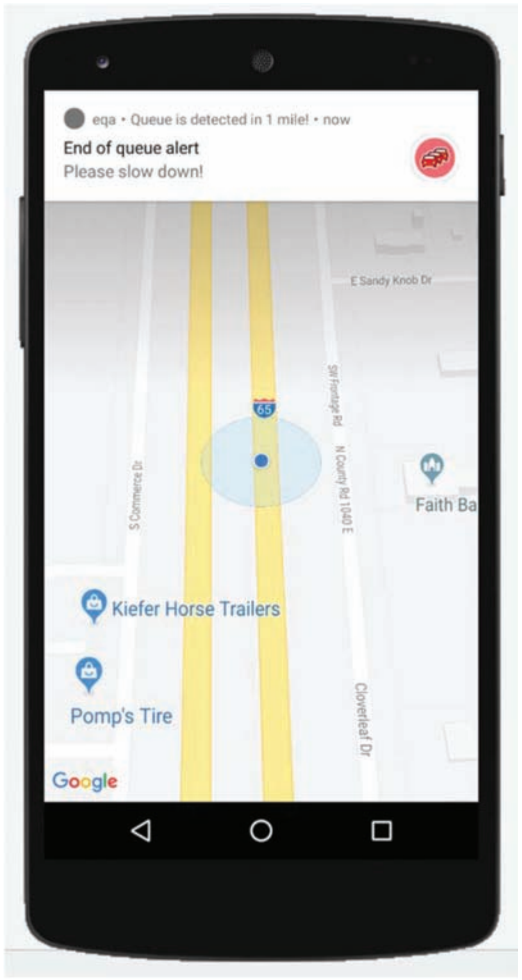

eqa • Queue is detected in 1 mile! • now
End of queue alert
Please slow down!

Figure 4.3 Notification on the smartphone.

particular device or program that other components expect to interact with. We can use the emulator to debug our program on the virtual environment without any need to have the real smartphone. Plus, we can test the application on different Android devices. Working with the emulator is a bit different. For example, we have to generate the GPS coordinates in a KML/GPS file and upload it into the emulator. Figure 4.4 and Figure 4.5 show the emulator setup in the Android Studio.

4.2.2 Hardware–Android Auto Device and Android-Based Smartphone

There are a variety of multimedia receivers that are applicable with the Android Auto function. The one we are using is Kenwood DDX9704S, which supports both Android Auto and Apple CarPlay. It has wide VGA color LCD display, HD radio, and Bluetooth. Touch panel control makes it easy operate. Dual phone connection is supported. When an Android phone is connected, an option for Android Auto will appear in the menu. The Android phone that we are using is a Samsung Galaxy S7 installed with Android v6.0 Marshmallow operating system. Octa-core processor,

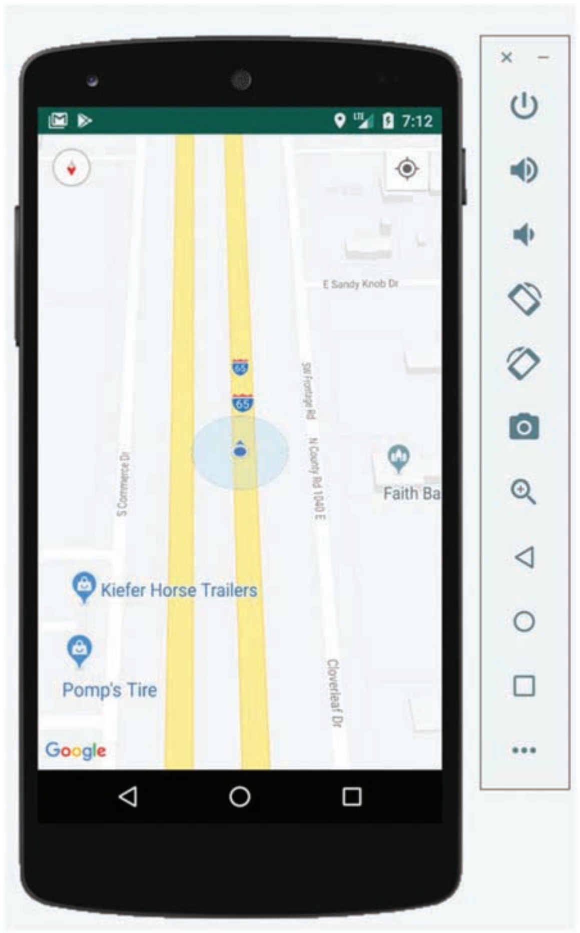


Figure 4.4 Android app emulator.

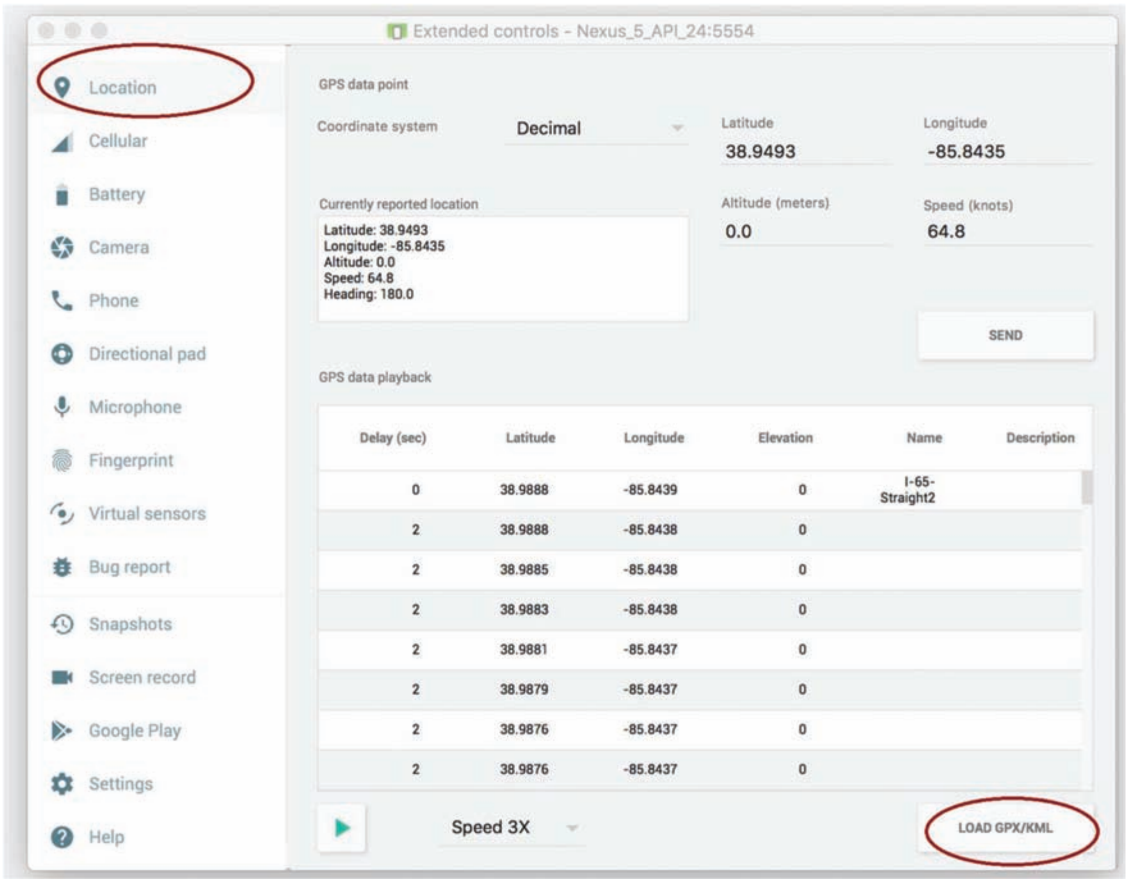
high-performance GPU, and 4GB RAM make the phone ideal for software development and evaluation. By touching the Android Auto icon on the menu, we are able to enter the Android Auto system. By default, the system could provide the functions include Google map, music players, and contact lists. More apps are accessible when they are installed in the connected Android phone. Figure 4.6 shows the Android Auto device and the Android phone that have been used in our testing.

The goal of using Android Auto is to issue alerts using a built-in vehicle hardware device. We used two ways to issue end-of-queue alerts. The first one is to use message/voice alerts. An Android app has been developed to implement this method. The app can run independently in the Android phone without Android Auto. But when the Android Auto is connected, the alarms and messages will all be transferred to the monitor of Android Auto. In this way, any Android phone can be used for Android Auto compatibility, with additional development. Figure 4.7 shows the interface of the message alert. The content in the banner and text card could be modified. Figure 4.7 shows the interface of the Android Auto display when the message alert is issued.

Besides, with the integration of Google Assistant, the alert messages can be converted into voice. Tapping on the banner or the text card, Google Assistant can read out the content and interact with the driver. The voice interaction can reduce the distraction level to drivers and achieve better safety while driving.

The alternative is to synchronize the Android Auto device with the connected Android phone and mirror the phone screen. As we found out, Android Auto can support more apps than only the default ones, but there is a limitation since most customized apps are forbidden by Google Inc. recently. The rooting of the phone is needed before mirroring the phone screen with the Android Auto device. In this way, more information can be shown including maps, images, and more intuitive interfaces. Figure 4.8 below shows the synchronized screens of the Android Auto device and the phone screen.

Both methods can issue end-of-queue alerts to drivers. However, they both have advantages and disadvantages. The method of message/voice alerts supports message reading and voice control and is also independent from the phone screen, i.e., the phone screen doesn't need to be turned on all the time. On the other hand, limited information can be provided as only texts are supported. Although the texts can be read through Google Assistant, it is still not very intuitive to use. Meanwhile, the method of synchronization of the phone and the Android Auto device shows the phone screen on the device monitor and is able to show much more information. Moreover, it is possible to control the alerts from both devices. At the same time, rooting the phone is a disadvantage because it has the risk of damaging the phone. In such case, these two solutions can be used for different needs.



Coordinate system: Decimal

Latitude: 38.9493

Longitude: -85.8435

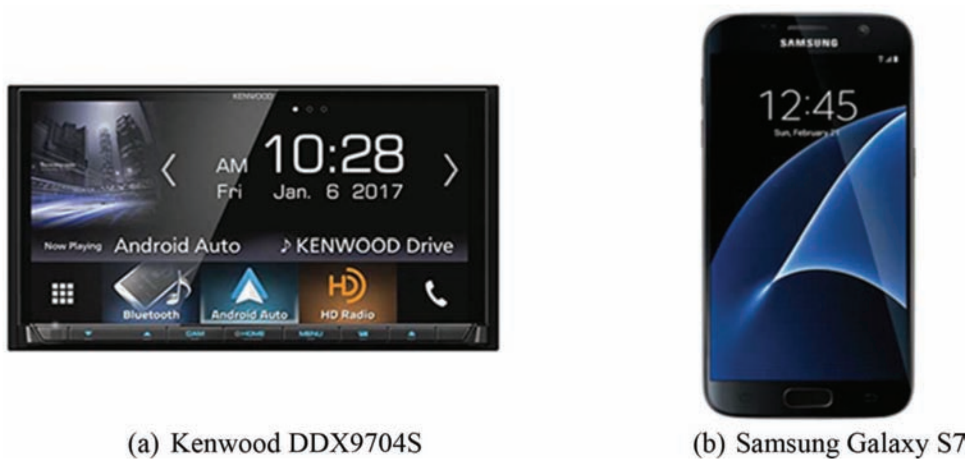
Altitude (meters): 0.0

Speed (knots): 64.8

Currently reported location
Latitude: 38.9493
Longitude: -85.8435
Altitude: 0.0
Speed: 64.8
Heading: 180.0

Delay (sec)	Latitude	Longitude	Elevation	Name	Description
0	38.9888	-85.8439	0	I-65-Straight2	
2	38.9888	-85.8438	0		
2	38.9885	-85.8438	0		
2	38.9883	-85.8438	0		
2	38.9881	-85.8437	0		
2	38.9879	-85.8437	0		
2	38.9876	-85.8437	0		
2	38.9876	-85.8437	0		

Figure 4.5 Upload KML files to emulator.



(a) Kenwood DDX9704S

(b) Samsung Galaxy S7

Figure 4.6 Android devices used for testing.

4.3 Summary

In this section, we summarized the hardware preparation and software development for the prototype end-of-queue alerting system. Software programs were developed to fetch the data from INDOT web service. Algorithms were developed to issue end-of-queue alerts. The Android Auto device and Android-based phone were investigated. Two methods for issuing alerts on these devices were developed. The advantages/disadvantages of the two solutions were evaluated.

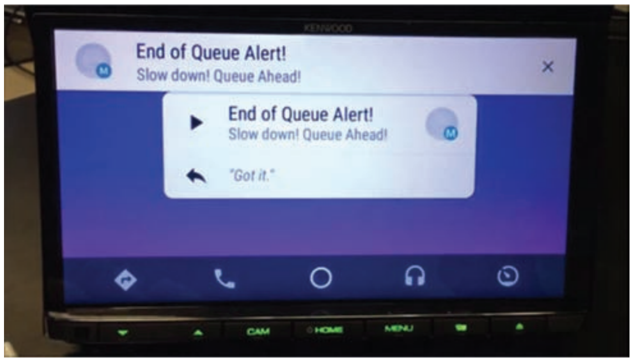


Figure 4.7 Message/voice alerts.

Figure 4.8 Synchronization of the smartphone and Android Auto device.

5. EVALUATION OF PILOT ALERTING SYSTEMS IN DRIVING SIMULATOR (TASK 4)

5.1 Introduction

In this task, a driving simulator-based study was conducted to evaluate the effectiveness of the developed prototype end-of-queue alerting system. Figure 5.1 shows TASI DriveSafety DS600c high-fidelity driving simulator at IUPUI. The driving simulator has 180 degrees of viewing angles to provide realistic and immersive driving experiences. A partial Ford Focus cab is instrumented with full-width front interior, standard driver controls, and active instrumentation. The driving simulator also supports different signs/visual signals assigned at different roadside locations.

At this stage of the project, the integrated data collection apparatus was first developed to prepare for all the follow-up testing tasks. Then a limited scale experiments were conducted to (1) test the data collection apparatus, (2) investigate the scenario development capability, and (3) study the effects of end-of-queue alerting system on driver performance and driving safety. With this purpose, five subjects were recruited for driving simulator-based testing. In different scenarios, driving performance data were collected and analyzed for the performance evaluation of the prototype system. This work will build the foundation to develop and conduct larger-scale experiments in the future to further improve the HMI, design, and performance of the end-of-queue alerting system.

For evaluating the pilot system, an experimental platform was designed and implemented. The goal of designing this platform is to simulate a driving environment in which the driver would be notified, through the mobile application, of the upcoming traffic hazards.

Since we already had a driving simulator, the objective in this phase was to make a bridge between the driving simulator and the mobile application. This bridge needs a staged server in between, which in this experimental platform would also mimic the INDOT server functions when connecting to the mobile application. Therefore, the mobile application receives simulated notifications through the staged server. In this section of the document, a staged server will be called server for short.

The platform and its functions are depicted in Figure 5.2, as three modules: HyperDrive (driving simulator), Server, and Mobile app. Of course, the connection between the server and the mobile application is through a wireless or 4G connection.

In this experimental platform, while a candidate person is using the driving simulator, location of the car, as well as other traffic information, are sent to the server. The server then provides the proper notification information, which will be pulled by the mobile application. Hence, while the driver is inside the driving simulator, his/her mobile phone would pull the simulated traffic information from the server and function accordingly.

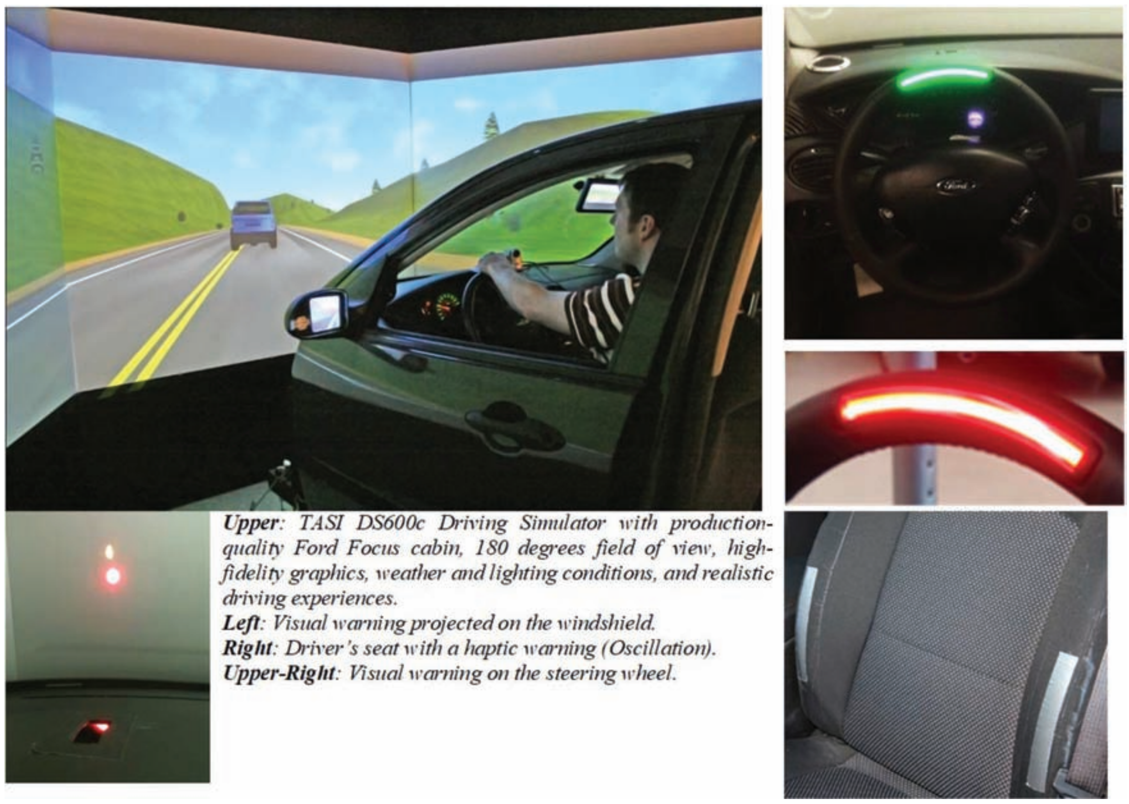
Given that we already had a driving simulator and a mobile app, the server and the connection between the other modules needed to be tailor-designed and implemented. In the next subsection, technical details are addressed and explained in detail.

5.2 Technical Approach

5.2.1 Building Modules for Connection and Communication

Implementation of the evaluation system consists of three interrelated modules. The first module is the networking details, around which the system design is revolving. The second module is the web service, providing the server and mobile application connection. And the third module is the connection and communication protocol between the server and the driving simulator. Details about all three modules are given as follows.

Module 1—Networking details. In order to avoid potential problems with the simulator machine that may cause it to disconnect from the internet, the server was set up on a separate machine. The connection between the simulator and server machine is through a



***Upper**: TASI DS600c Driving Simulator with production-quality Ford Focus cabin, 180 degrees field of view, high-fidelity graphics, weather and lighting conditions, and realistic driving experiences.*
***Left**: Visual warning projected on the windshield.*
***Right**: Driver's seat with a haptic warning (Oscillation).*
***Upper-Right**: Visual warning on the steering wheel.*

Figure 5.1 TASI DriveSafety high-fidelity driving simulator.

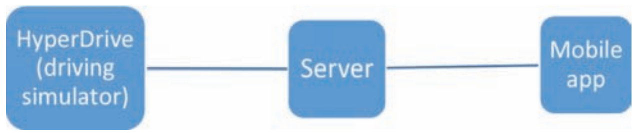


Figure 5.2 Building the experimental platform for subject testing.

directly connected Ethernet cable, where the IP addresses are manually set.

The connection between the server and the mobile application is established through a second network adapter (NIC) on the server machine, which is connected to the internet. The server machine on the latter NIC is given a static IP address to be accessed from outside. Only a few selected ports were left open on the server machine for incoming connections.

Module 2—Web service: Mobile app to server connection. The first step is to set up a web service. The mobile application would pull notifications upon request. The notification information is in JSON format, mimicking the INDOT server JSON files.

Since web servers and services are easier to set up in a Linux environment, the WSL on the Window OS was utilized to set up a Linux Ubuntu v18.04. Due to limited applications and for quick prototyping, the web service within the Linux OS was implemented using a Django development environment in debugging mode.

Currently, a webpage can be accessed through the address *134.68.77.18:1234* or *tasi.dynu.net:1234*. The *tasi.dynu.net* domain is acquired freely and for temporary use from the *dynu.com* website.

The notification can be obtained by the mobile application upon an HTTP request through the *tasi.dynu.net:1234/get-notif*. The notification information will be sent to the mobile application in JSON format.

The web service is designed to provide the mobile application with the simulated notification upon request. However, for a scenario where some fake notification needs to be manually generated, another service is accessibly designed at *tasi.dynu.net:1234/new-alert*. By visiting this address, a new notification will be generated which can be accessed by the mobile application. The fake notification information, for now, is hard-coded. In the future it might be possible to modify the fake notification information.

Module 3—Simulator to server connection. In this experiment, the information from the driving simulator needs to be sent to the server. According to the HyperDrive program documents, a socket connection could be made between the program as a client, and a receiving end as a server. Therefore, the design of this part is around network socket approach.

On the server machine, a server program was implemented. This server program is bound to a given port on the NIC with a direct network connection node, which has a manually set IP address. The server program would constantly listen and forward the incoming packets to the web service.

On the client-side, which is the driving simulator (HyperDrive program) we had some difficulty. Initially we attempted to make the socket connection between the HyperDrive program and the server work, which failed. The software documents were not very helpful in that regard. A response regarding this issue is still pending from the software developing company.

As a B plan, we decided to have the simulator information written in a file and develop an interface program on the client-side to read the file and prepare and send the information to the server. Currently this interface program on the client-side does the following:

1. Read the file (which is written into by the simulator program) line by line.
2. Parse each line and extract the required information from the file.
3. Prepare the information in JSON format to be sent to the server.

As of now, all the above functionalities were implemented in the client interface. However, the last bit of the puzzle is still holding us back from going forward. Unfortunately, the HyperDrive simulator writes the simulated information into the file only at the end of the simulation, not during runtime. What we expected was to have the information dumped into the file during the simulation and then send it to the server. This way, while the candidate driver was driving the car, the mobile application would have pulled the information from the server.

5.2.2 Building the Test Scenarios Using the Driving Simulator

There are three components in the driving simulator, which are HyperDrive, Vection, and Dashboard. The HyperDrive is the preparation and scenario creation stage. The Vection is the software that runs the simulation. The dashboard is the software that interfaces the HyperDrive and Vection. The overall structure of the driving simulator is shown inFigure 5.3.

In order to test the highway end-of-queue alerting system with real driving conditions on I-465, a test scenario with a simulated I-465 was built in the driving simulator DS-600. The primary software was HyperDrive, whose purpose is to build the test environment with roadway networks and controlled entities. Entities are used to determine an object in the world, such as a vehicle or a pedestrian.

As shown in Figure 5.4, the total length of I-465 is about 52.79 miles (≈84.46 km) with a width of 1.71 miles and length of 14.68 miles. There are three main purposes to build the I-465 test scenarios: (1) the simulator can represent real-life vehicles within the simulation by using GPS data in the future; (2) enable validation of developed theories with real vehicle data; and (3) allow vehicles with realistic data from sensory hardware to interact with other virtual vehicles. Therefore, a ring highway test scenario was built based on the real I-465 in Figure 5.4.

The HyperDrive Interface is depicted in Figure 5.5, the position of all toolbars are also marked, such as Menu Bar, Main Toolbar, Visualization Window Toolbar, and Authoring Palette. The solid white lines are world boundaries. White dots are spaced in a 200 meters grid in the Visualization Window. The Authoring Palette contains all of the available Tiles, Scenario Tools, Entities, and the World Object Browser. The specific tabs and their corresponding elements are summarized as follows:

1. **Tile Tab:** It contains all the tiles available to users, different culture (a type of environment such as urban and rural), road type (such as freeway or community road), and signals (such as traffic lights) can be selected as desired requirements. The two ways with six lanes were selected to simulate the I-465. Since there are some limitations of the elements, we cannot change the curvature of curved road elements in the simulator. Thus, for some curved road in real I-465, we can only use the general curved road provided in the Tile Tab.
2. **Scenario Tools Tab:** It contains all tools that help define the events in the scenario such as where the subject starts and at what location an event is triggered. Three starting points were used in the scenario with respect to the three driving scenarios of normal driving (ND), driving with distraction (DWD), and driving with drowsiness (DWDS). Since we have six test scenarios based on the scenario design (ND with or without alert, DWD with or without alert, and DWDS with or without alert), the distances from the starting point to the end of queue for three driving scenarios of ND, DWD, and DWDS are around 10 miles, 15 miles, and 20 miles, respectively. Several location triggers (based on the required distance from the trigger) were also utilized to trigger the event, which can let other vehicles join the highway with ego vehicle and maintain the desired speed to make a traffic jam at the end of the test scenarios. Since we cannot add the traffic queue and issue a warning dynamically when the simulation is running due to the limitation of the

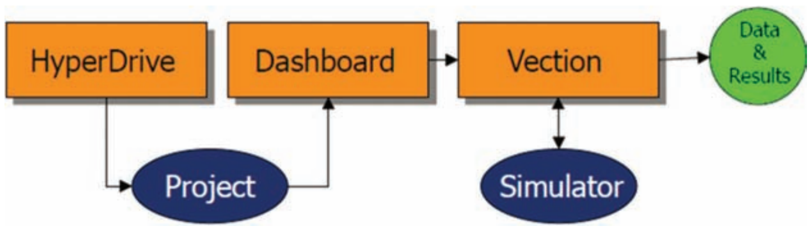


Figure 5.3 Overall structure of the driving simulator.

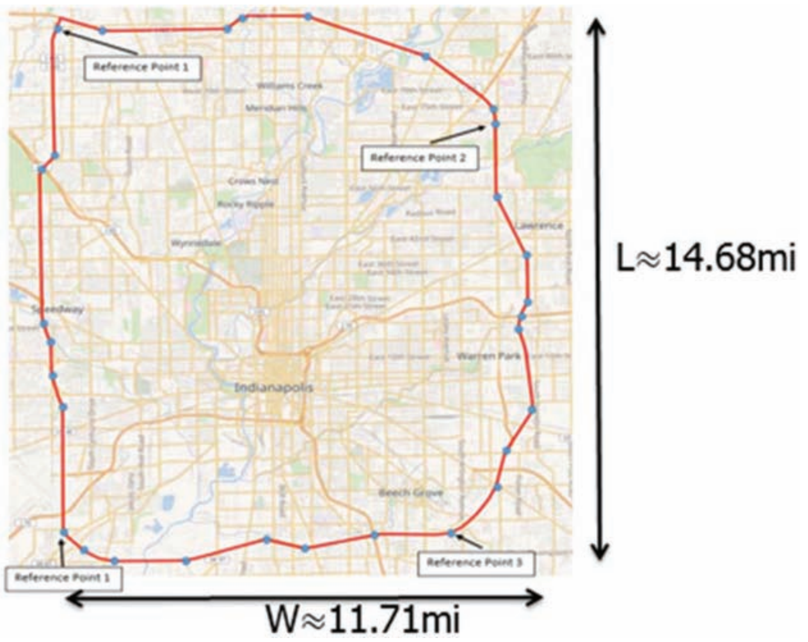


Figure 5.4 Illustration of highway I-465.

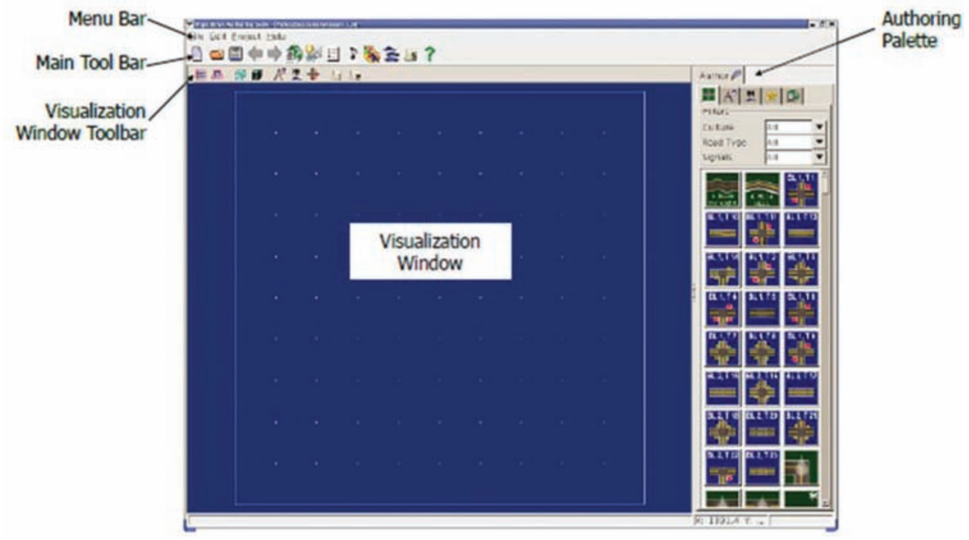


Figure 5.5 HyperDrive interface.

driving simulator, three indicators of buildings for triggering the alerts in cellphone were designed.

3. **Entities and Static Entities Tab:** It has three types of entities—dynamic entities, kinematic entities, and static entities. Several dynamic entities of vehicles were used and they can be controlled by scripting to simulate the queue. Some static entities of buildings were also placed on the roadside of Highway I-465 in the driving simulator, which makes the driving conditions more realistic.
4. **World Object Browser:** Once all the elements have been added to the scenario, every object in HyperDrive has a set of properties that are accessed in the World Object Browser. From here, the position of X, Y, Z of each object can be modified based on the specific design purpose.

The completed I-465 highway test scenario is demonstrated in Figure 5.6, and the test condition in our driving simulator room is shown in Figure 5.7. The vehicle dynamic information was also displayed on the left projected screen while testing, which includes vehicle longitudinal acceleration, vehicle lateral acceleration, steering angle, X position, Y position, and Z position on the map. Table 5.1 also shows several key variables that were collected after the simulation.

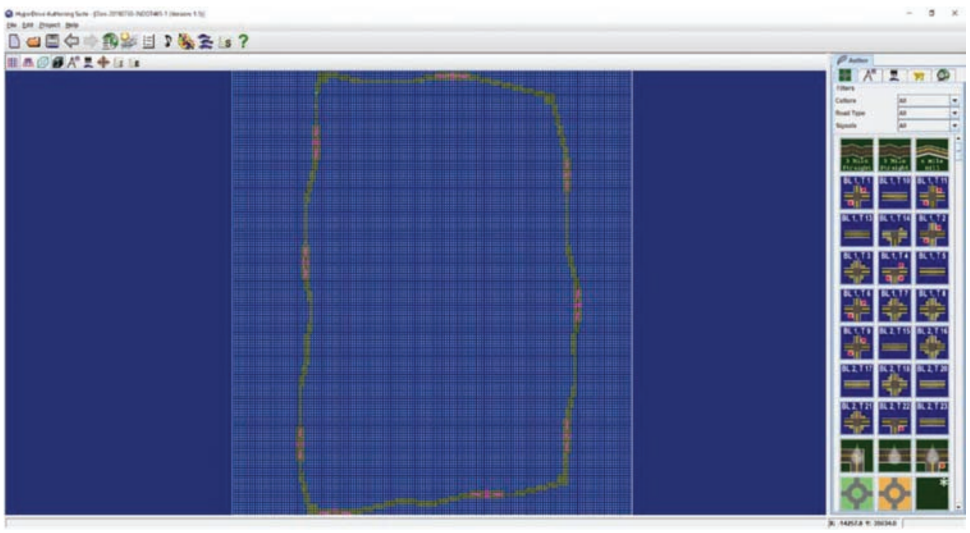


Figure 5.6 Completed highway I-465 in DS-600 driving simulator.

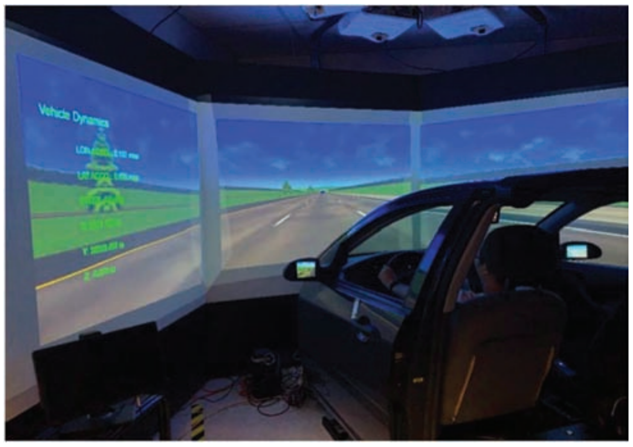


Figure 5.7 Actual test conditions.

TABLE 5.1
Key Variables Collected After Simulation

Key Variable Name	Unit
Longitudinal acceleration	m/s^2
Crash or not	0 or 1(binary)
Crash speed	m/s
Brake pressure	0–100%
Steering angle	degree
Vehicle location	m

5.2.3 Design of the Data Collection

Based on the current development of the end-of-queue alerting system, we have designed six tests to evaluate the effects of having an end-of-queue alerting system in the driving simulator. The six scenarios are categorized as either no-alert, no end-of-queue alerting system in the car, or alert, an end-of-queue alerting system in the car. If an end-of-queue alert is issued, the alert will be issued about 1.5 miles before the end of the queue. Under both categories, there are three driving environments: normal, distracted, or drowsy. The non-alert category will act as a baseline to be compared with the tests in the alert category. To ensure authenticity, subjects are required to follow all driving laws and rules observed in the real world. The detailed descriptions of the environments are the following:

- **Normal:** During this test, subjects are asked to drive in the simulator for about ten miles on a replica of highway I-465 with moderate traffic. At the end of the ten-mile drive, a traffic queue will appear. This test takes about ten minutes.
- **Distracted:** During this test, subjects are asked to drive in the simulator for about 15 miles on a replica of highway I-465 with moderate traffic. Before the subject drives past the would-be trigger point in non-alert scenarios or triggers an alert in alert scenarios, the data collector will randomly send the subject a couple of text messages that require responses. After the subject drives past the would-be trigger point or a traffic alert is issued, the data collector must message the subject to create a distraction to the upcoming traffic. At the end of the 15-mile drive, a traffic queue will appear. This test takes about 15 minutes.
- **Drowsy:** During this test, subjects are asked to drive for about 20 miles on a replica of highway I-465 with moderate traffic. The subject should drive as they normally would, but they should be drowsy when they perform this test. To help ensure drowsiness, this test is the last environment tested in both categories. Additionally, the time of this test is performed is planned around when subjects will naturally get tired (i.e., in the afternoon after lunchtime). This test takes about 20 minutes.

The total time it takes to collect data for all six tests is about two hours. The order of the tests is shown in Figure 5.8.

To measure the effects of an end-of-queue alert system, the maximum braking pressure, crash rate, and other factors are collected and analyzed. The analyzed

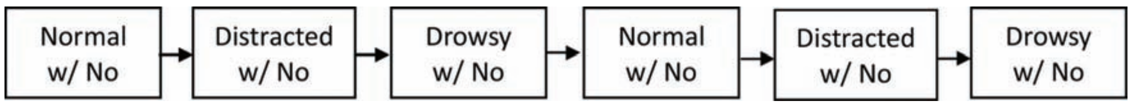


Figure 5.8 Order of tests for data collection.

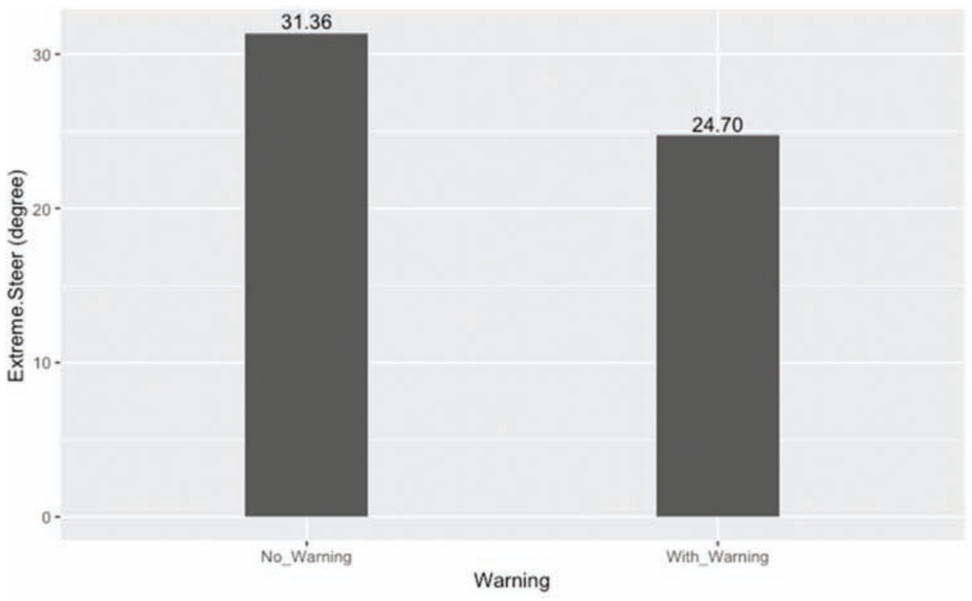


Figure 5.9 Comparison of extreme steering angles between cases with and without end-of-queue alerts when approaching the end-of-traffic queues.

data are collected after the car passes alert triggering locations (no matter if the alert is really triggered or not). This means that no matter if the case is with alerts or without alerts, for the same scenarios, the data were collected for the same part of the driving route.

5.3 Results

Five subjects, all under the age of 35, were recruited to participate in this pilot study. The analysis focuses on evaluating the usefulness of the prototype end-of-queue system in reducing extreme behaviors when approaching to the traffic queues. We would like to compare the driver inputs (brakes and steering wheels) as well as the vehicle dynamics measures when the alert is present or not for normal, distracted, and drowsy drivers.

Due to the limited sample size, there are no actual crashes observed during the driving simulator study.

5.3.1 Effects of Alerts on Braking, Steering Wheel, and Vehicle Dynamics

The first part of the data analysis is to compare the extreme driver inputs and extreme vehicle longitudinal and latitude acceleration when approaching the end-of-queue with and without the alerts. The results are shown in Figures 5.9 to 5.12. We can tell that for all measurements, including extreme steering angles, extreme braking percentage, extreme deceleration (longitudinal deceleration), and extreme side-way acceleration (latitude acceleration), the average values are smaller for cases with end-of-queue alerts provided. This indicates that this alert system can overall reduce intensive driving behavior, and thus has the potential to increase driving safety.

5.3.2 Effects of Alerts and Driver Status on Driving Performance

Although the previous section has confirmed the benefits of applying the end-of-queue system, it is also interesting to see what the effects are when considering both driver status and the presence of alerts as combined variables. The results are shown below.

The first discussion focuses on the extreme steering wheel angles, representing the driver's input to the steering wheel when approaching the end of queue with or without alerts. Different driver statuses are also applied in the analysis. The results are shown in Figure 5.13. We can see that for drowsy and normal driving, the alert can help remove extreme driver steering inputs. However, for the distracted driving scenario, drivers on average, have a bigger extreme steering angle when alert is provided compared to the distracted driving without alert, although the difference is limited. This result may be caused by the limited number of sample size. Also, more detailed time-series data analysis needs to be conducted to further reveal the braking behavior patterns.

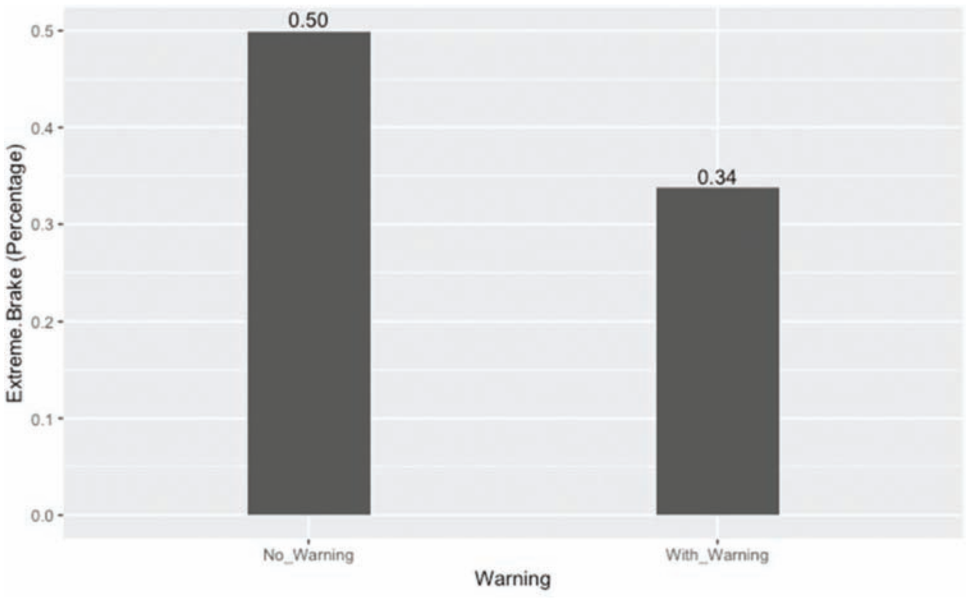


Figure 5.10 Comparison of extreme braking percentage between cases with and without end-of-queue alerts when approaching the end-of-traffic queues.

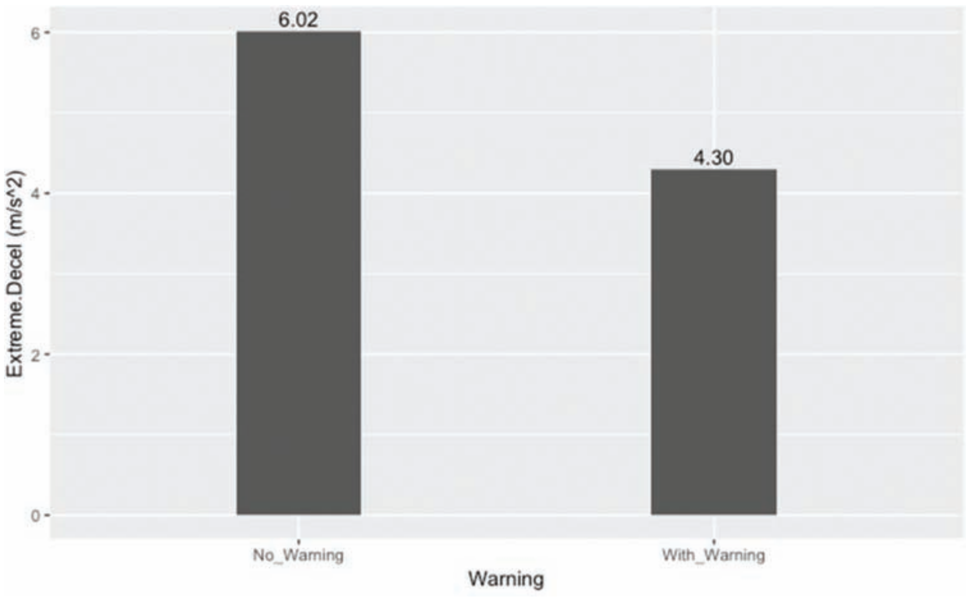


Figure 5.11 Comparison of extreme deceleration between cases with and without end-of-queue alerts when approaching the end-of-traffic queues.

Similar results were found for the extreme braking percentages, among cases with and without alerts and under different driver status, as illustrated in Figure 5.14. For this measurement, end-of-queue approaching alerts can significantly reduce the average extreme braking percentages for drivers in normal or distracted status. However, for drowsy drivers, the effects of the alerts on this braking measure is reversed. This again may suggest the necessity of increasing the sample size and conduct more detailed time-based data analysis to better understand the behavior patterns.

5.4 Summary

In this section, we mainly introduced the driving simulator DS-600 and how we built the I-465 highway test scenarios. The background and basic descriptions of DS-600 have been introduced. The functionality of the driving simulator is also demonstrated along with its embedded software and hardware. Several test scenario examples of DS-600 and its overall structure were also demonstrated. The specific tabs, tools, and entities used for creating the I-465 test scenarios were

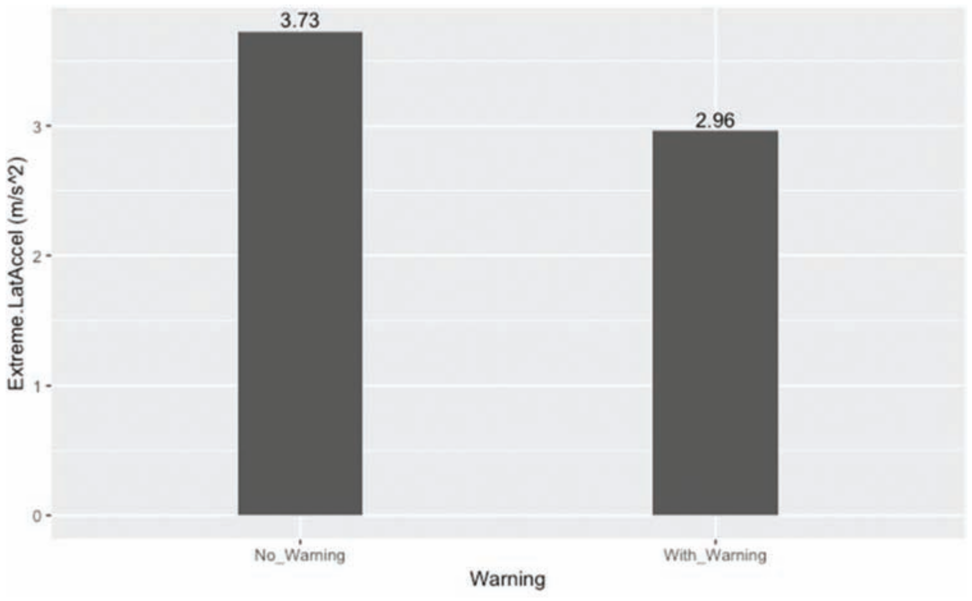


Figure 5.12 Comparison of extreme latitude deceleration between cases with and without end-of-queue alerts when approaching the end-of-traffic queues.

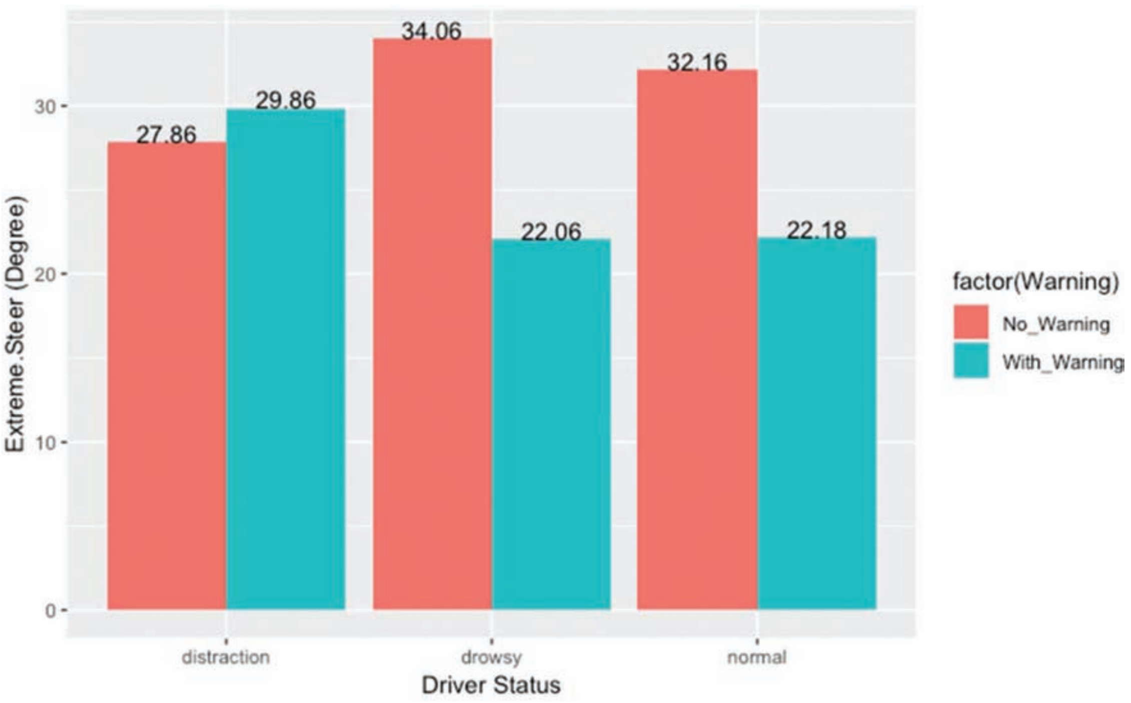


Figure 5.13 Comparison of extreme steering angles between cases with and without end-of-queue alerts when approaching the end-of-traffic queues at different driver statuses.

also illustrated, and the desired data were saved after the simulation. Five subjects were recruited to perform driving tests on the driving simulator for pre-defined test scenarios. Data were collected and analyzed. The results showed the benefits of having the end-of-queue alerting system to improve driving safety.

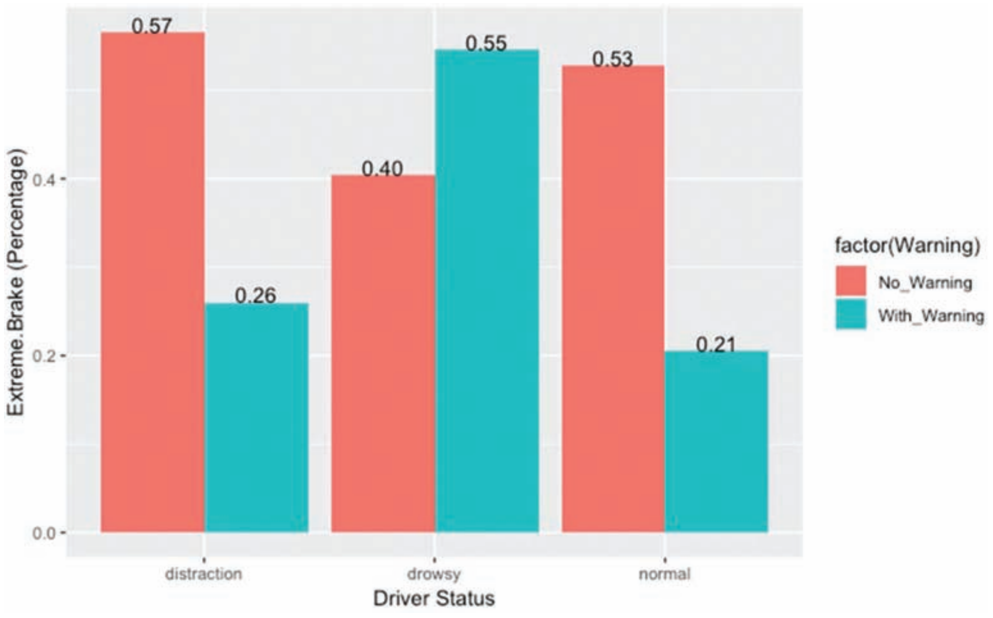


Figure 5.14 Comparison of extreme braking percentages between cases with and without end-of-queue alerts when approaching the end-of-traffic queues at different driver statuses.

6. EVALUATION OF REFINED PILOT ALERTING SYSTEMS WITH LIMITED ROAD TESTING (TASK 5)

To test the system in real traffic situations, a group of tests was performed on different highways. The developed end-of-queue alerting app was installed on the smartphone of the testing subjects, which continuously collect location and highway traffic queue data. Testing this application in a predefined scenario was a hard task because of the real-time nature of INDOT data. Since the data is real-time, we had to develop a mechanism to detect the delta speed events in order to drive there for our testing. We added extra icons on the map to see all the queue events across the city. We also added related information on the icons, which are shown in Figure 6.1.

We went on different highways where a delta-speed event was detected and, in most cases (around 80%), we were able to get the alert successfully, as shown in Figure 6.2. These tests are necessary to find problems in the computer programs and improve them for the app applications. The application performs well on straight paths in most of the cases. Occasionally, we have problems of issuing alerts on a very curved path. For the cases we didn't get the notifications, we simulated the same situation on the emulator and debugged the code. In general, the application had a good performance in most cases on highways and we are still doing more tests to find and solve any possible problems.

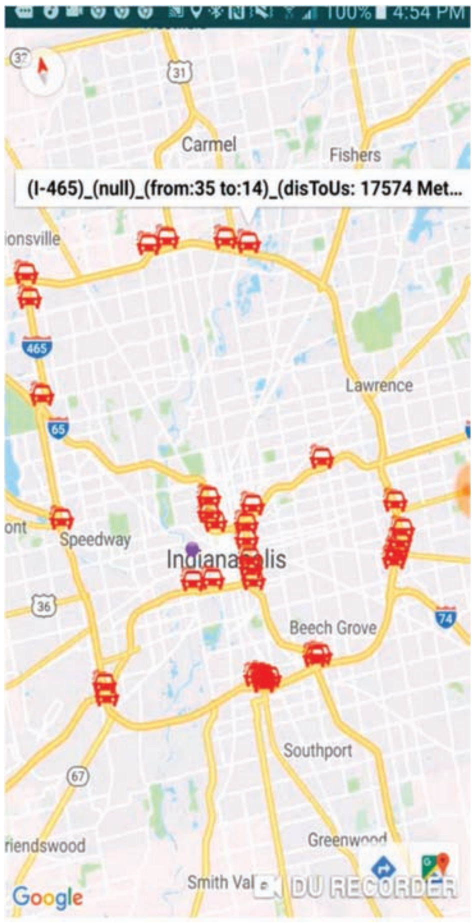

(I-465)_(null)_(from:35 to:14)_(disToUs: 17574 Met...

Figure 6.1 Visualizing INDOT speed events map.

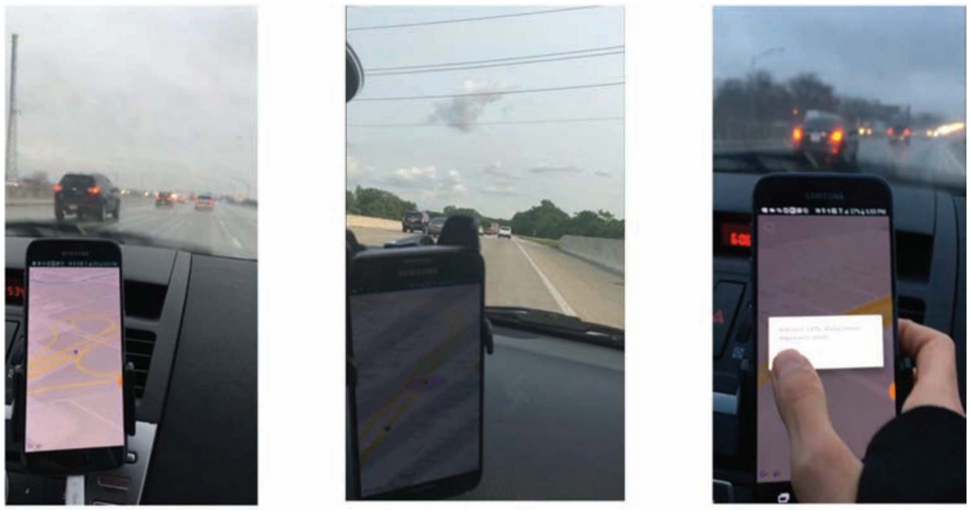


Figure 6.2 On-road tests on different highways.

7. SUMMARY

In this project, we develop a prototype in-vehicle end-of-queue alerting system that is based on the probe vehicle data from INDOT. Speed changes among different road segments were used to identify slow traffic queues, which are compared with vehicle locations and moving directions to detect potential end-of-queue crashes. This prototype system was designed to issue alert messages to drivers approaching the highway traffic queues via an Android-based smartphone app and an Android Auto device. The performance of this system was evaluated using both the driving simulator and on-road test runs. The results showed the effectiveness and benefits of this prototype system to improve driving safety.

REFERENCES

Lee, J. D., McGehee, D. V., Brown, T. L., & Reyes, M. L. (2002). Collision warning timing, driver distraction, and driver response to imminent rear-end collisions in a high-fidelity driving simulator. *Human factors*, *44*(2), 314–334. https://doi.org/10.1518/0018720024497844

Mekker, M., Li, H., McGregor, J., Kachler, M., & Bullock, D. M. (2017). Implementation of a real-time data driven system to provide queue alerts to stakeholders. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems*. Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/itsc.2017.8317648

Mekker, M., Remias, S. M., McNamara, M. L., & Bullock, D. M. (2020). *Characterizing interstate crash rates based on traffic congestion using probe vehicle data* (JTRP Affiliated Reports, Paper 31). https://docs.lib.purdue.edu/jtrpaffdocs/31/

Nowakowski, C., Vizzini, D., Gupta, S. D., & Sengupta, R. (2012). Evaluation of real-time freeway end-of-queue alerting system to promote driver situational awareness. *Transportation Research Record: Journal of the Transportation Research Board*, *2324*(1), 37–43. https://doi.org/10.3141/2324-05

Tampère, C. M. J., Hoogendoorn, S. P., & van Arem, B. (2009). Continuous traffic flow modeling of driver support systems in multiclass traffic with intervehicle communication and drivers in the loop. *IEEE Transactions on Intelligent Transportation Systems*, *10*(4), 649–657. https://doi.org/10.1109/TITS.2009.2026442

About the Joint Transportation Research Program (JTRP)

About This Report

An open access version of this publication is available online. See the URL in the citation below.

Li, L., Chen, Y., Tian, R., & Li, F. (2019). *Back of queue warning and critical information delivery to motorists* (Joint Transportation Research Program Publication No. FHWA/IN/JTRP-2019/24). West Lafayette, IN: Purdue University. https://doi.org/10.5703/1288284317102