Fall 2017

SenSys: A Smartphone-Based Framework for ITS applications

Abdulla Ahmed Alasaadi
Old Dominion University

SENSYS: A SMARTPHONE-BASED FRAMEWORK FOR ITS APPLICATIONS

by

Abdulla Ahmed Alasaadi
B.Sc. February 2003, University Of Bahrain, Bahrain
M.Sc. 2005, Lancaster University, England

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
December 2017

Approved by:

Tamer Nadeem (Director)

Kurt Maly (Member)

Michele Weigle (Member)

Mecit Cetin (Member)

ABSTRACT

SENSYS: A SMARTPHONE-BASED FRAMEWORK FOR ITS APPLICATIONS

Abdulla Ahmed Alasaadi
Old Dominion University, 2018
Director: Dr. Tamer Nadeem

Intelligent transportation systems (ITS) use different methods to collect and process traffic data. Conventional techniques suffer from different challenges, like the high installation and maintenance cost, connectivity and communication problems, and the limited set of data. The recent massive spread of smartphones among drivers encouraged the ITS community to use them to solve ITS challenges.
Using smartphones in ITS is gaining an increasing interest among researchers and developers. Typically, the set of sensors that comes with smartphones is utilized to develop tools and services in order to enhance safety and driving experience. GPS, cameras, Bluetooth, inertial sensors and other embedded sensors are used to detect and analyze drivers' behavior and vehicles' motion.

The use of smartphones made the data collection process easier because of their availability among drivers, the set of different sensors, the computation ability, and the low installation and maintenance cost. On the other hand, different smartphones sensors have diverse characteristics and accuracy and each one of them needs special fusion, processing, and filtration methods to generate more stable and accurate data. Using smartphones in ITS faces different challenges like inaccurate readings, weak GPS reception, noisy sensors and unaligned readings. These challenges waste researchers and developers time and effort, and they prevent them from building accurate ITS applications.

This work proposes SenSys a smartphone framework that collects and processes traffic data and then analyzes and extracts vehicle dynamics and vehicle activities which can be used by developers and researchers to create their navigation, communication, and safety ITS applications. SenSys framework fuses and filters smartphone's sensors readings which result in enhancing the accuracy of tracking and analyzing various vehicle dynamics such as vehicle's stops, lane changes, turn detection, and

accurate vehicle speed calculation that, in turn, will enable development of new ITS applications and services.

ACKNOWLEDGMENTS

Firstly, I would like to express my sincere gratitude to my advisor Dr. Tamer Nadeem for his continues support, guidance, patience, and motivation during my Ph.D study.

My sincere thanks also goes to my thesis committee: Prof. Kurt Maly, Dr. Michele Weigle and Dr. Mecit Cetin, for their insightful comments and encouragement.

Also, I want to show my gratitude to Prof. Hussain Abdulwahab who passed away last December for all his help, care, and guidance during my time at Old Dominion University.

Last but not least, I would like to thank my family for supporting me in every way and throughout my Ph.D study and my life in general.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Transportation is getting growing attention because of its impact on economy, environment and people's health. A recent United States Department of Transportation (US-DOT) report "2015 Transportation statistics annual report" [1] revealed that traffic congestion wastes around 3 billion gallons of fuel in 2014, and nearly 6.9 billion extra delay hours. A Texas A&M Transportation Institute (TTI) report [5] titled "The 2015 urban mobility scorecard" predicted that the nationwide delay will jump from 6.9 billion to 8.3 billion hours in 2020 and the total cost of congestion will jump from $160 billion to $192 billion in 2020. On average, drivers spend over 40 hours stuck in traffic each year, while drivers on America's 10 worst roads spent 84 hours or 3.5 days a year on average in gridlock. The problem becomes more severe in areas with high population where vehicle commuters experienced an average of 63 hours of extra travel time. In addition, the total cost of the traffic accidents reached about $794 billion dollars in 2012 and 45.7% of these accidents are speed related. Figure 1 shows the total wasted fuel as a result of congestion since 1985.

The issues that face the transportation system can be summarized as the following:

- Mobility: Drivers mobility is affected by the number of hours wasted on congestion and accidents. For example, drivers wasted 6.8 billion hours in congestion in 2014.
- Environment: It has been well documented that the main cause of today's air quality is the emissions generated by road transportation [8, 9, 10]. The studies also showed that, road congestion is not the only threat to economic growth but also a main contributor to global warming. Carbon emission is caused by fuel consumption and the number of gallons wasted. In 2014, more than 3 billion gallons were wasted because of congestion only.

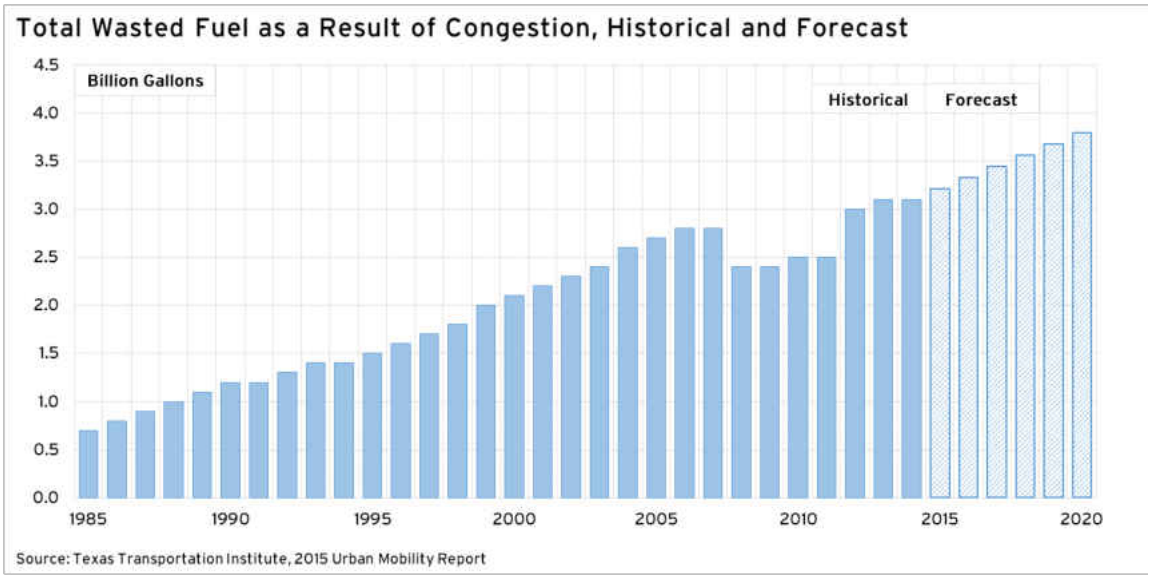


FIG. 1: The total wasted fuel as a result of congestion [5]

- Economy: The cost of the fuel and time wasted on congestion affects the economy. The total cost of congestion was 160 billion dollars in 2014, and the cost of traffic accidents in 2012 reached 794 billion dollars.

- Safety: According to the National Highway Traffic Safety Administration (NHTSA), more than 32,000 people died and more than 2.3 million were injured in motor vehicle traffic crashes in 2013. Table 1 shows the number of injuries and fatalities caused by transportation from 1990 to 2013.

1.1 INTELLIGENT TRANSPORTATION SYSTEMS

The Unites States government supports the development of technologies to reduce the total cost of traffic congestion by improving safety on roads, reducing severity of congestion, fuel consumption, carbon emission, traffic accidents, and travel delays. Intelligent Transportation Systems (ITS) utilize various methods of data collection and traffic monitoring using different technologies like inductive loop detectors, video detection systems, and magnetometers to monitor and understand the traffic. ITS systems utilize different technologies to collect traffic data from roads, drivers, and vehicles. The collected data helped in finding solutions for many ITS challenges such

TABLE 1: Injuries and fatalities caused by transportation [1].

Injured Persons by Transportation Mode	1990	2000	2009	2010	2012	2013
Air General aviation	409	309	273	256		
Highway	3,230,666	3,188,750	2,217,000	2,243,000	2,360,000	2,313,000
Railroad	22,736	10,424	7,227	7,376		
Transportation Fatalities						
Air General aviation	770	596	478	450		
Highway (in vehicle and non-occupants)	44,599	41,945	33,883	32,885	33,782	32,719
Railroad	729	631	544	601		

as designing better traffic light control systems, parking guidance and car navigation systems. ITS plays an important role in addressing environmental problems by collecting and analyzing traffic data which can be used to control traffic regulations which lead to reducing the emission produced.

Many applications developed using different ITS technologies, for example, safety applications like the emergency vehicle notification system [11] which detects and reports incidents in accurate and timely manner. Another example is the traffic enforcement camera, which monitors, detects and identifies vehicles that offend the traffic rules like over-speeding or crossing red light. One of the safety applications used in modern trucks is the lane departure system [12] that notifies the driver when he changes the lane without noticing. Other projects developed to help drivers in reducing the carbon dioxide emissions like the "Driving Change" project that provides feedback to drivers to change their driving behavior [13]. This project used by employees of Denver's city government in USA, and they were able to change their driving styles to reduce 10% carbon dioxide emissions [13].

Challenges and Problems of ITS systems

Existing ITS Systems face different kinds of problems that need to be addressed:

- Privacy: Several data collection technologies like surveillance cameras and electronic tolling can track users and raise privacy concerns among drivers. Many countries like Canada restrict the location of surveillance to only public areas and specifies a specific zoom level that cannot be exceeded.

- Cost: The cost of the intelligent transportation system depends on the technology used, and it varies from technology to another. Most existing technologies require the installation of sensors and devices in roads, the equipment could be too expensive in some cases. One of the most significant elements in the choice of detection technology is the life-cycle cost. The total cost can include the installation costs, maintenance costs, traffic control, driver delay and related additional fuel consumption, and additional pavement costs in case of new pavement needed during installation and maintenance of some detectors.

- Deployment and maintenance: Most road detectors suffer from poor reliability due to improper installation. Many detectors like inductive loops need to be installed during road construction; the installation requires a saw cut to the pavement and stopping the traffic during the time of installation and maintenance. Because of the cost and the installation difficulty of installing new ITS technologies, the authorities install these technologies in few selected places.

- Limited Data: The data collected from different technologies are limited to the types of sensors used in that technology, for example, the data collected by inductive loop detectors are vehicle passage, presence, count, and occupancy and the data collected by cameras are limited to footage, plate numbers, traffic density, and speed detection.

1.2 SMARTPHONES IN ITS

Traffic challenges are encouraging the ITS community to look for new technologies that can assist or replace the current systems. The high technology used in smartphones and their rich features in terms of processors, sensors, communication and memory encouraged the researchers to utilize them in enhancing the existing ITS systems. US-DOT released the 2045 plan [1] about the future of ITS, the report expected the smartphones to be an important factor in ITS because of their rich sensors and accessories. Smartphones use has increased sharply in recent years, and

therefore more people are using them in their daily activities. According to AT&T 2014 sales report, 94% of their phone sales were smartphone devices [14]. This surge in the number of smartphone users attracted developers to develop more than 1.3 million apps in Google Play [15] store and 1.2 million apps in Apple App Store as of July 2014 [16]. This large spread of smartphones made it easy to deploy new low cost technologies to collect data from drivers and vehicles. Current smartphones come with different types of sensors like, inertial sensors, camera, microphone, and GPS. The data derived from these sensors encourage researchers to develop different types of applications in many domains like sports, health, localization, smart homes, and ITS. ITS applications such as safety and navigation applications which utilize GPS, OBD, and inertial sensors to study road traffic, driver behavior, and track vehicle's dynamics.

The existing smartphones based ITS applications use phone either for data collection or to do one specific application. Developers face difficulties in developing smartphone ITS applications because of the difficulty in processing the noisy data produced by phones inertial sensors. More advanced smartphone-based ITS applications can be developed if there is a framework that does all the data processing and the feature extraction. This motivated us to build a reliable smartphone based ITS framework that reads, processes, filters the raw data and extracts different features that can be used by developer to easily build different types of ITS applications. This framework will be explained in detail in Chapter 3.

1.3 SENSYS FRAMEWORK

Smartphones can play important role in solving ITS challenges but they have not been utilized completely by developers because there is no tool to help them to create ITS applications. This motivated us to build an efficient framework (SenSys) that can be used by developers to develop low cost and efficient ITS applications on smartphones. The SenSys framework will save time, money, effort and resources by providing a set of services that collects, filters, processes, analyzes and extracts the data. This framework can always accommodate the new methods and algorithms to extract driving and road features. Developers can use SenSys to utilize multiple sensors and fuse their data to build efficient and accurate ITS applications. SenSys framework scalable and expandable where new filters, services and data processing algorithms can be added and integrated to the framework. SenSys framework will

enable developers to use multiple sensors at the same time, they can also choose what sensors to use based on their application requirements and hardware limitations. SenSys will have three layers, the first layer is responsible for collecting, fusing, filtering sensors' data. The second layer utilizes the filtered data and applies different algorithms and optimization methods to extract vehicle's motion dynamics. The third layer is responsible for delivering the extracted features to the applications. SenSys will provide a framework to be used by ITS smartphones developers with a simple interface which hides behind it many filtering, fusing, features exaction and optimization algorithms.

1.4 CONTRIBUTION

This thesis contributes the following:

- Design and development of SenSys framework that can be used by smartphone developers to build ITS applications. No such framework is available today.
- Design and development of different algorithms to filter and fuse different sensors to correct the faulty readings and extract informative data [17] [18].
- Design and development of a new method to align smartphone's coordinates with vehicle's coordinates [17].
- Provide a set of APIs that can be used by smartphone developers to build ITS applications.
- Evaluation of SenSys framework through developing a set of applications:
  - Design and development of a new and accurate method to identify a vehicle's parking spot using smartphone's inertial sensors only [19].
  - Design and development of a new method that identifies vehicles in the same lane using smartphone's inertial sensors only [20].
  - Design and development of a new method that detects vehicle's number of axles and class type using smartphone's inertial sensors only.

1.5 ROAD MAP

The remainder of this thesis is organized as follows: in Chapter 2 we provide a detailed background and literature review that goes through the existing works on intelligent transportation system and the use of smartphones to solve the ITS problems and enhance the traditional methods in collecting and analyzing traffic data. In Chapter 3 we introduce the SenSys framework and its components. In Chapter 4 we explain the data collection and preparation methods. In Chapter 5 we explain the components of the basic and advance vehicle dynamics extraction module and in Chapter 6 we describe the set of APIs provided by the SenSys framework. Chapters 7,8,9, and 10 explain the ParkZoom, InLanCom, GoGreen, and the vehicle type detection applications. Chapter 11 includes the conclusion and the future work.

CHAPTER 2

LITERATURE REVIEW

2.1 INTELLIGENT TRANSPORTATION SYSTEMS

Most of the conventional and current techniques in intelligent transportation systems are focusing in traffic data collections. Popular methods like inductive loops, magnetometer, video image detection systems and others have their own challenges and advantages which will be discussed thoroughly in this chapter.

2.1.1 INDUCTIVE LOOP DETECTORS

The most popular technology used in traffic data collection is the Inductive loop detectors (ILD) (Figure 2). ILDs consist of one or more loops of wire embedded in the pavement and connected to a control box. When a vehicle passes over or stops on the loop, the inductance of the loop is reduced showing the presence of a vehicle [21, 22]. Inductive loops can detect vehicle count, occupancy and traffic flow. The data collected by inductive loops can be analyzed and used for congestion and incidents detection. ILD can be unreliable and perform poorly because of the improper installation which also leads to high maintenance (life-cycle) cost and inaccurate detection. When installed and maintained properly the ILD will work in all weather and light conditions and can be a very accurate detector in terms of vehicle counting.

2.1.2 MAGNETOMETER

Another popular technology used in traffic data collection is the magnetometer. The magnetometer is an inroad sensor that detects the magnetic disturbances in the earth's field as a vehicle passes over. The magnetometers are only able to detect vehicle presence, they are effective for counting vehicles and detecting rates. The problem with this technology is the limited number of applications it can be used in because of the limited data it provides. The magnetometer can be installed in places where inductive loop detectors cannot be installed like bridges and the installation

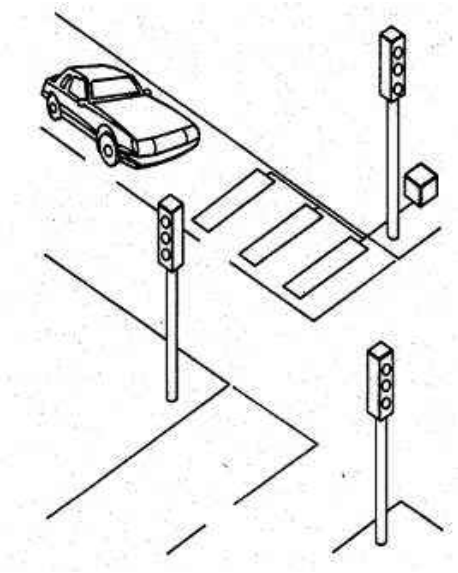


FIG. 2: ILD diagram.

requires drilling a hole into the pavement.

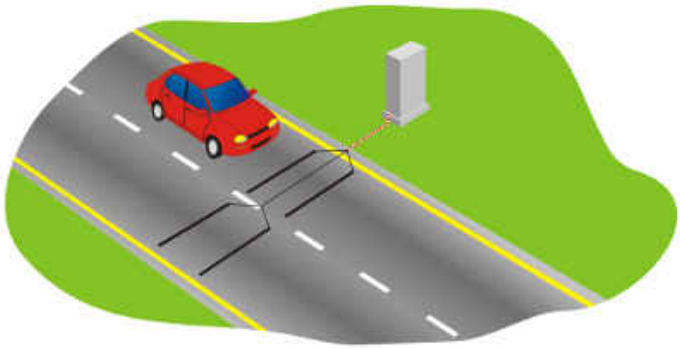


FIG. 3: Piezo electric detectors.

2.1.3 PIEZOELECTRIC DETECTORS

Piezoelectric detectors (Figure 3) come in form of metal strips placed on road surface. The piezoelectric detectors can transform the mechanical energy into electrical energy each time a vehicle wheel passes over them. Such technology is used for vehicle counting, classification and weighting. The data collected using this technology is limited to vehicles' detection and counting [23].

FIG. 4: Pneumatic tubes.

2.1.4 PNEUMATIC TUBES

Pneumatic tubes (Figure 4) are hollow rubber tubes placed over the roadway where vehicle counts or speeds are needed. It is connected to a box that analyzes the pulses to count vehicles and estimate their speeds. They are used for temporary data collections because they can be removed and placed easily on road surfaces but they can get easily damaged [23].

2.1.5 VIDEO IMAGE DETECTION SYSTEMS

Video Image Detection Systems (VIDS) uses computer vision technologies to analyze traffic data collected with video cameras. For example, [24] uses image processing techniques to detect vehicles in videos. VIDS systems are used to monitor roads conditions [25], control signals, detect incidents [26], and classify vehicles [27, 28]. Most of the video networks used today by ITS community are using analog video systems because they can get high quality videos in a relatively low cost. They can cover wide areas with small number of cameras which make them efficient in detecting density, queue lengths, and speed profiles. VIDS are used by law enforcement authorities to detect license plates. Detecting license plates in two points can be used for speed estimation and travel time calculation. The initial price of the VIDS hardware and software is much higher than the other conventional methods like inductive loop detectors. On the other hand, the installation and maintenance cost is lower in case of video systems. The analog video systems are more used than the digital video

systems because of their high price which is three times more expensive than the traditional digital systems. Weather conditions such as rain, snow, fog, or dust can affect the quality of the video image and the performance of the system [29].

2.1.6 INFRARED AND LASER SENSORS

Active infrared sensors transmit low-energy laser beams to a target area on the pavement and measure the time for the reflected signal to return to the sensor. The presence of a vehicle is measured by the corresponding reduction in time for the signal return [23]. However, weather conditions and lens clearance can affect the performance of the infrared sensors.

2.1.7 RADAR SENSORS

Radar sensors use a continuous signal to determine the time delay of the return signal, thereby calculating the distance to the detected vehicle. Radar sensors can detect volume, presence, classification, speed and headway of vehicles. However, radar sensors can experience dead detection zones and ghost vehicles when installed in areas with barriers, fencing, or other obstructions [23].

2.1.8 MICROWAVE SENSORS

The microwave sensors transmit electromagnetic waves to detect the presence of vehicles. The deployment of microwave sensors is easier than ILD and magnetic sensors because they can be installed on top of traffic lights. On the other hand, these sensors get affected by the interference from nearby microwave devices [30].

2.2 SMARTPHONES IN ITS

Smartphones sensors and computation ability attracted the ITS community to develop ITS apps. The use of smartphones sensors in transportation has been discussed in many papers, many of them use the GPS like the APT project (Accurate Pedestrian Tracking) [31] which uses the GPS to track outdoor pedestrian, while others use the inertial sensors, microphone [32], camera [33], or Bluetooth to communicate with the on-board diagnostic device that connects the smartphone with vehicle's computer like [34]. Usually, developers use one or more of the sensors based on the application's requirements.

GPS is the most popular sensor used by developers to build ITS applications because of its simplicity and the rich information it provides. Many navigation, safety and communication ITS applications use GPS in smartphones like Cai et al. [35] which uses the GPS only to find the optimal driving speed to avoid red traffic lights. Other projects combined the GPS readings with readings from other sensors like Fazzen et al. [36] which uses the GPS and the accelerometer to detect hazardous driving behaviors. Ruta et al. [37] uses GPS, OBD (On-board diagnostics) and the web to build a knowledge-based real-time car monitoring system. The problem with GPS is the high power consumption when compared to other sensors and its weak signals in tunnels and near tall buildings.

Another major method of data collection used by many ITS applications is the OBD which can be connected by Bluetooth to access the information provided by vehicles computer. The work of Choi et al. [38] and Shaout et al. [39] used smartphone to communicate with the OBD to analyze and classify driver's behavior. Their systems use the speed to detect the unsafe driving behavior. OBD data also used in assisting the driving behavior to make it more environment friendly like the Artemisa project [40] which encourage drivers to reduce their fuel consumption by showing them their instant fuel consumption. Smartphones and OBD are also used for car diagnosis like OBDDoctor[41], Torque[42] and many other applications which communicate with the on-board computer and read the warnings codes. OBD tend to provide instant and accurate data, but it is limited to the data provided by the on-board computer which is not enough to detect all vehicle dynamics and driving behaviors.

Smartphone cameras have been utilized in many projects using computer vision algorithms, for example, SmartLDWS project [43] uses smartphones camera to create a lane departure warning system which imitates the built-in systems that come with some newer cars and trucks. SmartLDWS uses the camera to detect the lines between lanes and alarm the driver if he passes over these lines. Singh et al. [44] used the camera in detecting the unintended maneuvers. In addition to the previous examples, cameras are used in other applications that intended to assist the drivers like Ling et al. [45] that recognizes traffic lights and Munoz et al. [46] which calculates the optimal decelerating pattern to stop in traffic light. The common problem with camera based applications is that the performance drops in inclement weather conditions and bad lighting.

2.2.1 INERTIAL SENSORS IN ITS

Smartphones' inertial sensors value has been recognized by motion detection systems, driver behavior systems and navigation systems. For example, WreckWatch [34] uses inertial sensors to detect accidents in real time, Thompson et al. [47] and the MotoSafe project [48] detect accidents and provide situation awareness to emergency responders. WreckWatch and MotoSafe use inertial sensors to detect the shock caused by the accident.

Because of the inertial sensors ability to read shocks caused by vehicles movements, they have been used on motion mode detection. For example, Bedogni et al. [49] classifies the motion mode into car mode and train mode based on the accelerometer readings. Hemminki et al. [50] used accelerometer readings to detect the transportation mode, the study was able to differentiate between car, train, bike, waking an running modes based on the readings variance.

Inertial sensors also used in road and traffic monitoring, for example, NeriCell [51] uses smartphone inerital sensor and GPS to monitor road traffic conditions, and Eriksson [52] used mobile sensor network in monitoring road surface. Inertial sensors readings can be processed and used for feature extraction. Three types of features can be extracted regardless of the signal source: statistical-based features like mean, and variance, time-based features like integrals, and frequency-based features like FFT and wavelet [50]. Transportation mode detection approaches can mix and match the previous features to extract road specific ones. For example, segment-based features that characterize a road section based on patterns of acceleration and deceleration that defines how the road looks like, as stated in NeriCell project [51] or how different motion modes behave as Hemminki et al. discussed in their paper [50]. Road structure creation is another use for smartphone sensors in ITS. Bumps and potholes detection depends on detecting low speed zones, then using a high-pass filter to remove low frequency components, and finally detecting peak acceleration in the gravity direction [52, 53]. For road surfaces monitoring, Strazdins et al. [54] used pothole detection algorithms over the cloud servers to monitor road surfaces.

One of the main problems that face smartphones developers is aligning the device's coordinates with the vehicle's coordinates. This problem is explained in detail in Chapter 4. The readings of the sensors are referenced to the phone's frame not the vehicle's frame. Therefore developers created applications that force drivers to fix the orientation of the phone as the Parkzoom project [19], Fazeen et al. [36] and

Eren et al. [55]. Fazeen et al. [36] designed an Android application that uses the accelerometer readings to analyze hazardous driver behaviors, but they had to fix the phone in a fixed orientation where the phone's Y axis is pointing toward the front and the screen is facing the roof. The problem with this solution is that, it forces the driver to fix the phone on the windshield with specific orientation, and it prevents the driver from using or moving the phone from its original orientation.

Other solutions used application specific solutions like Aldunate et al. [56] which process the inertial sensors to satisfy their applications' requirements, the problem with their solution is that it is not generic and cannot used by other applications. Almazan et al. [57] used the GPS to get the moving direction, but GPS is not the optimal solution in aligning the readings references because of the power consuming, low reading rate, and unavailability near high buildings or during cloudy weather. SenSys framework uses inertial sensors based solution to handle this problem but the option of using the GPS or other sensors is still available to users.

There are other solutions that worked on other domains, like estimating the heading of a walking person. For example LaneQuest [58] estimated the heading of a walking person using principal component analysis (PCA) and sensor fusion to detect phone orientation and heading direction by using the pendulum movement of the user's arm to find the direction of motion. Another work by Kunze et al. [59] proposed a method that estimates the orientation of the phone when it is in the users pocket while walking. Developers of Padati [60] used the gyroscope and magnetometer to recognize users' activity while phone is located in their pocket. Morales et al. [61] went an extra step to calculate the acceleration in the motion direction using PCA; this increased the accuracy in detecting users' activities. However, all these works consider the periodic movements of the human, which is not the case for transportation.

Unlike other works, SenSys aligns the readings by using the accelerometer readings caused by the vehicle's acceleration and deceleration on the direction of motion to estimate the vehicle's heading. The data preparing module takes the readings and maps them to the Earth coordinate to remove the gravity accelerations and then apply the PCA in the two dimension plane. Since the PCA works well with people movement it needs extra processing and filtering to make it work with vehicles which will be done in the SenSys framework.

TABLE 2: Related works that worked with the orientation problem.

Project	GPS	OBD	Inertial Sensors	Ordination	Domain
Vehicle Dynamics[62]	No	No	Yes	Limited scenarios and performance	Vehicular
Drunk driver detection [63]	No	No	Yes	Fixed	Vehicular
CityDrive [64]	Yes	No	Yes	Flexible, needs GPS	Vehicular
Safe driving [36]	No	No	Yes	Fixed	Vehicular
Patrol[52]	No	No	Yes	Fixed	Vehicular
Wreckwatch [34]	No	Yes	Yes	Flexible, needs OBD	Vehicular
Nericell[51]	Yes	No	Yes	Flexible, needs GPS	Vehicular
Apt [31]	Yes	No	Yes	Flexible, need GPS	Vehicular
Parkzoom[19]	No	No	Yes	Fixed	Vehicular
SenSpeed [65]	No	No	Yes	Fixed	Vehicular
Projects [58] [59] [60]	Yes	No	Yes	Flexible	Human
UniCoor	No	No	Yes	Flexible/ Generic	Vehicular

Smartphones sensors also used in vehicles localization like finding a vehicle in a large parking lot. There are Smartphone apps like Find My Car app [66] and MyCar Locator [67] which use GPS to locate cars in big parking lots. Such applications have a large demand by people based on their pages on Google Play store where each app is installed more than 500,000 times. Although the existing solutions are popular, they come with set of problems. First they depend only on GPS, so if GPS signal is weak then the app is not going to work and this applies to all indoor parking lots. Second the accuracy of the current outdoor applications goes up to 50 meters away from the exact location [68]. Patwari et al. [69] targeted the indoor localization by using sensors' network to find the relative location of a sensor or object to the other sensors, but such solution need a pre-fixed framework and the accuracy depends on the number of prior sensors installed. Our solution is different, it can work indoor and outdoor, and it uses smartphones only with no extra equpments installed on

vehicles or parking-lots and it can identify the parking spot taken by the veihcle.

Inertial navigation systems (INS) have been used in locating moving objects like ships, airplanes, submarines, robots [70, 71], and humans. Such systems use inertial sensors with other inputs to estimate the correct location for a moving object. Davidson et al. [72] combined the low cost INS with the GPS to estimate the location of a car in places where the GPS reception is weak. In times when the GPS goes off, the INS is used to trace the movement of the vehicle, but this will cause an accumulated error in calculating the speed from an accelerometer reading. Systems in [72] use the GPS to correct the readings periodically using the Kalman filter while other systems use the vehicle speedometer like Georgy et al. and Iozan et al. [73, 74] to get the speed, and uses INS to detect turns and other activities. Reinstein et al. [71] uses special sensors placed on the wheel to detect the speed, and uses INS to detect turns. For contentious positioning, Zaho et al. [75] used the Extended-Kalman filter to update the inertial sensors with the GPS readings periodically.

SenSys framework uses inertial sensors to extract vehicles movement which can be used by developers to estimate vehicle's location without using the GPS or other external sensors. Each project in the existing solution discussed in the previous sections are designed and built to solve a specific problem. Unlike others, SenSys is a generic framework that can be used to build smartphones applications for ITS. Conventional ITS solutions are expensive to install, maintain, and give limited information, while SenSys is easy to use and access solution that can be used to build low cost, accurate, and high performance ITS solutions. Table 3 summarizes some of the existing ITS smartphone applications.

TABLE 3: Existing Smartphones ITS applications.

Project	Sensors Used	Summary
CityDrive [64]	GPS,INS, Crowed-source	Uses the information collected from other smartphones suggest proper speed for drivers so that they arrive at intersections in green phase. It needs a server to collect and process the data.
Safe driving [36]	GPS,Acc.	Detecting hazardous driving behaviors. The phone has to be in a fixed orientation
Patrol [52]	GPS,GSM,INS	Using a mobile sensor network for road surface monitoring. The phone has to be in a fixed Oriantation
Wreckwatch [34]	OBD, INS	Accident detection and notification
Nericell [51]	GPS, INS	Monitor road traffic using GPS
Apt [31]	GPS	Outdoor pedestrian tracking, suffers from low accuracy in urban areas
MotoSaf [48]	GPS, INS	Detect high risk maneuver for motorcycle's riders
SMaRTCar [43]	OBD,GPS	Uses OBD and GPS for data collection, and traffic monitoring. Connected to Arduino board.

CHAPTER 3

SENSYS FRAMEWORK

3.1 OVERVIEW

This chapter will describe the SenSys framework design and its layers. The framework is divided into three main layers, the data collection and preparation layer, features' extraction layer, and the application interface layer. Figure 5 shows an overview of the framework. The first layer is the "Sensors data collection and preparation layer" which reads the raw data directly from sensors and sends them to different modules for processing and filtration. The second layer is the "Vehicle Dynamics Extraction layer" which extracts two types of features, basic features that represent the basic movement of a vehicle and the advanced features that represent the advanced movements. The third layer is the "Application Interface layer" which contains different types of applications that can be built on top of the previous two layers and provide useful information for developers that can be used on various domains. All these layers and modules will be explained in more detail in this chapter.

3.2 DATA COLLECTION AND PREPARATION LAYER

The data collection and preparation layer collects and prepares the data to be ready for analysis and feature extraction. It contains two types of modules, the data collection modules which are responsible for collecting real-time data from the various sensors and the data filtering and preparation modules which filters and prepares the data for further processing. The data collection modules communicate directly with different sources, OBD, GPS, and phones sensors. The data collected will be sent to the data preparation modules where the raw data will be filtered and processed to be used by the Vehicle Dynamics extraction layer.

3.2.1 DATA COLLECTION

Modern mobile phones come with a variety of sensors that can be utilized to analyze and understand the activities in the surrounding environment. This module

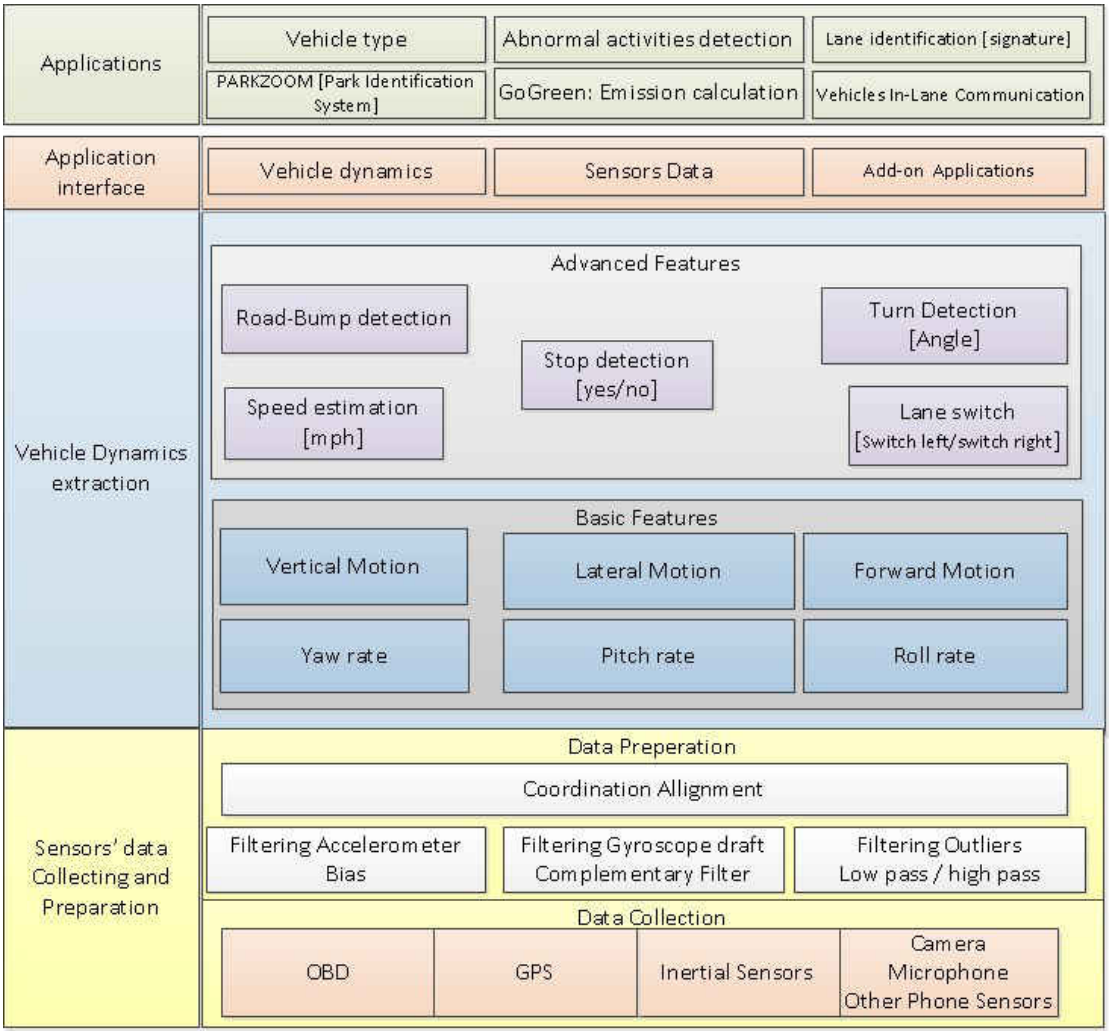


FIG. 5: SenSys Framework.

manages the coordination between available sensors and the framework, and sets the sensors configuration.The smartphone sensors used in the Data Collection module are inertial sensors, GPS, OBD, camera, microphone and other sensors. In case of new sensors and technologies, the framework enables new modules to be added to this layer. The rest of this section will explain in more detail the list of sensors used for data collection and how they work.

The first type of sensors are the inertial sensors. Most modern smartphones come with a set of inertial sensors (accelerometer, gyroscope and magnetometer). These sensors can be used to understand phone's movements, fixing screen's orientation, and in games and different types of applications. Inertial sensors sense the motion over the phone 3-axes. Figure 10 shows the axes of the phone in the local frame.

Accelerometer

The first and most used inertial sensor is the **accelerometer** which measures the acceleration forces along phone's local frame axes. These forces could be dynamic forces caused by moving the phone, or static like the gravity constant force. Accelerometers used in smartphones are small Micro Electro-Mechanical Systems (MEMS) designed to be sensitive only to one direction on the plane. Modern smartphones combine 3 accelerometers perpendicular to each other to measure the acceleration on the three dimensions. Figure 6 shows a 2-axis accelerometer. The accelerometer measures the time rate of change of velocity in m/s^2. The data generated from the accelerometer is vector having three axes (x,y,z). The vector's readings represent two types of accelerations: Gravity acceleration which points toward the center of Earth with a value of 9.81 m/s^2, and Other accelerations caused when moving or vibrating the phone.

Gyroscope

The second inertial sensors is the **Gyroscope** which is a device for measuring the angular rotational velocity. 3-axis gyroscope generates three readings; each reading is the angular velocity that indicates how fast the phone is rotating around a particular axis. Gyroscope used to detect the change in phone's orientation and the spin movement. Unlike accelerometers, gyroscope is not affected by gravity.

Accelerometers can be effected by vehicle's motion. Figure 8 shows the vertical accelerometer readings while the smartphone was placed on different vehicles.

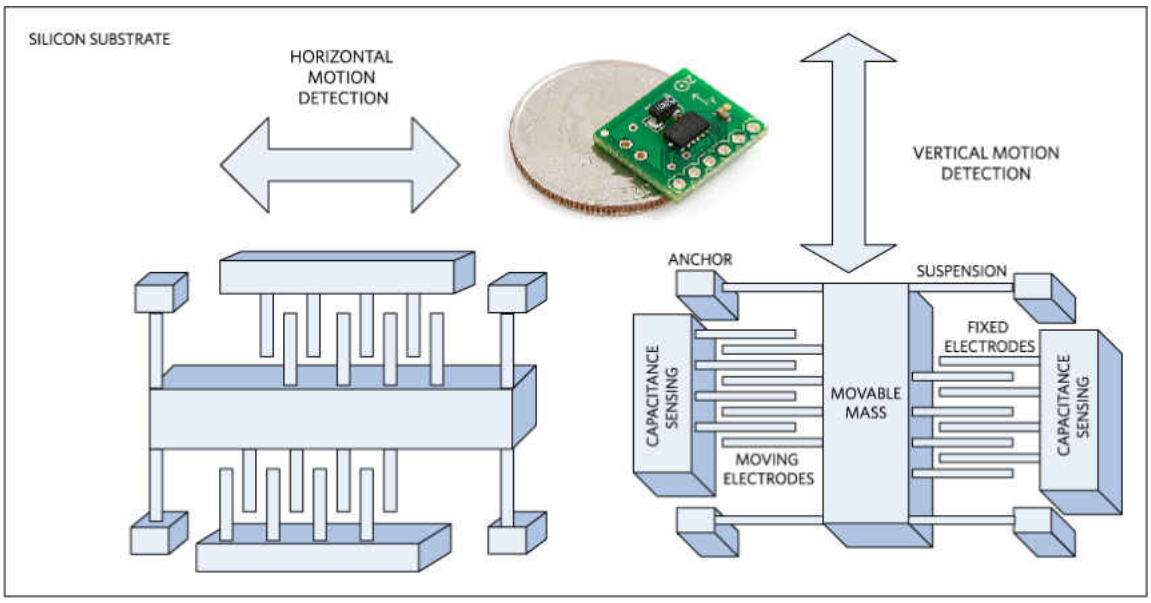


FIG. 6: A two-axis accelerometer [6]

The traditional mechanical gyroscope in Figure 7 consists of a spinning wheel mounted on two gimbals which allow rotation along the three axes. Smartphones use MEMS gyroscopes, Micro-Electro-Mechanical Systems (MEMS) is a technology used to create tiny integrated devices or systems that combine electrical and mechanical components. Compared to traditional gyroscopes, MEMS gyroscopes have smaller sizes, lower costs, higher precision and easier to integrate. There are different types of MEMS gyroscopes that are using different technologies, like the Tuning Fork Gyroscopes, Vibrating-Wheel Gyroscope, Wine Glass Resonator Gyroscopes, and Foucault Pendulum Gyroscopes. Gyroscopes that come with smartphones return three values per reading, each value represent the change of rate around one of the axes. The values returned are in Radians/second. To calculate the total angle of rotation, we need to integrate the values over the time taken.

Magnetometer

The third inertial sensor that comes with most smartphones is the **Magnetometer (Compass)**. Magnetometer is a micro-electro-mechanical device for measuring magnetic field. Magnetometer sensors are made using different approaches like Lorentz force based MEMS sensor, Electron Tunneling based MEMS sensor, and

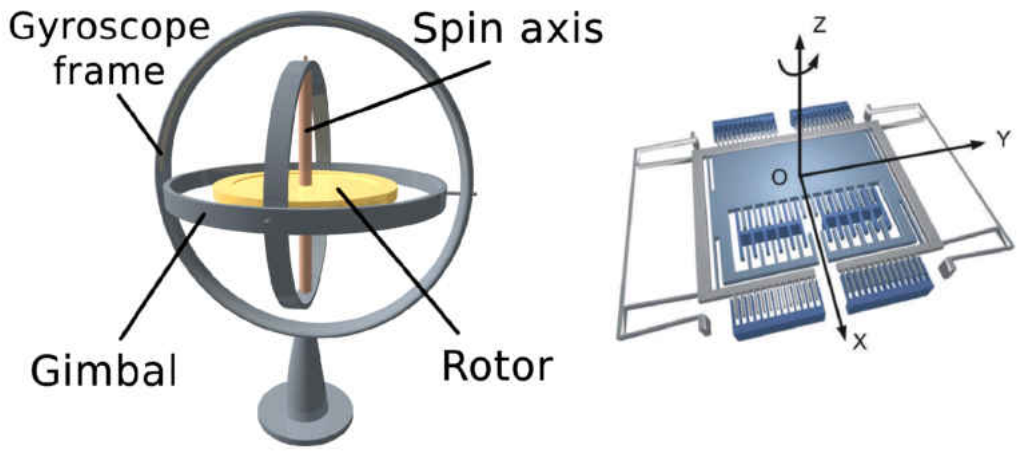


FIG. 7: Traditional Gyroscope vs MEMS Gyroscope [6]

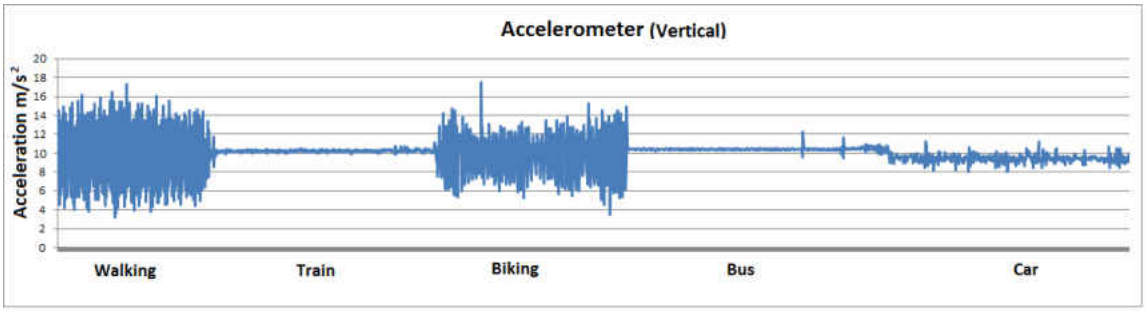


FIG. 8: Using accelerometer to detect driving modes.

MEMS compass. MEMS magnetometer sensors are smaller in size than the traditional magnetometers so they can be integrated to many devices. Moreover, MEMS magnetometers have a very low price comparing to traditional magnetometer, which enable including it in most devices. The magnetometer readings are a vector of the measurements of the component of the magnetic field in a particular direction, in reference to the device's local frame orientation. Three-axis magnetometers use three orthogonal sensors to measure the components of the magnetic field in all three dimensions. The data returned from the magnetometer readings are in micro-tesla.

Global Positioning System (GPS)

The second type of sensors used in this framework is the **Global Positioning System (GPS)**. GPS chips come with most modern smartphones. The GPS receiver

needs a clear line of sight with three GPS satellites to calculate the 2-D location (latitude and longitude), and 4 different GPS satellites to compute the 3-D location of the phone (latitude, longitude and altitude). The GPS unit can use the computed user's position to calculate the speed, bearing, trip distance, distance to destination and more. GPS unit are very useful in smartphones because it can determine user's location with one second sampling rate, calculates and provides other information such as speed and bearing. It is available in most smartphones and used in many types of applications.

But the GPS units are known to have some drawbacks that affect their accuracy and reliability. For example, GPS needs line of sight with 3 or 4 different GPS satellites to be able to calculate the current location. In smartphones, the average location calculation error is around 10 meters, which can be not reliable for many applications. Moreover, GPS does not work properly near tall buildings, trees, inside tunnels, or indoors. In Android smartphones, the GPS unit updates user's location every one second, and based on that the GPS speed, bearing, and time is updated every one second.

On-board diagnostics (OBD)

Another sensor that is used in transportation is the **On-Board Diagnostics (OBD)**. In the 1970s and early 1980s manufactures started to use electronic tool to diagnose engine problems and to control engine functions. The On Board Diagnostic systems standards have been developed and resulted in the OBD standard which introduced in the 1990s. This new standard granted more engine control and the ability to monitor many devices in the car. In 1996 the OBD standard is made mandatory for all cars manufactured in the United States. In addition to identifying malfunctions within the vehicle the OBD provides a real time data about the vehicle current status.

In SenSys, the data-collecting module establishes a Bluetooth connection between the phone and the OBD device. The module sends set of queries to the OBD to establish and configure the connection and to retrieve real-time vehicle's data in an agreed rate. OBD enables users to query the on board computer for specific information, the query includes the Parameter ID (PID). The standard defines many PIDs, but the manufacturers also have many PIDs specific to their vehicles. Table 4 shows some of the standard OBD PIDs.

TABLE 4: Description for some of the standard OBD requests

PID	Bytes	Description	Min	Max	Unite	Formula
0A	1	Fuel pressure	0	765	kPa	A*3
0C	2	Engine RPM	0	16,383	rpm	((A*256)+B)/4
0D	1	Vehicle Speed	0	255	km/h	A
0F	1	Intake air temperature	-40	215	C	A-40
11	1	Throttle position	0	100	%	A*100/255
1F	2	Run time since engine start	0	65,535	seconds	(A*256)+B
31	2	Distance traveled since codes cleared	0	65,535	km	(A*256)+B

Other sensors

In addition to all the previous sensors, modern smartphones come with variety of sensors like camera, light sensor, proximity sensor, barometer, camera and microphone. This framework is built to be expendable and reliable, so it can accommodate new sensors and include more features.

The **Camera** is an important sensor that is used by variety of applications. ITS systems used cameras for a longtime in traffic-lights scheduling, and lane departure systems. Typically, cameras are mounted on structures above or adjacent to the roadway. Cameras in smartphones can be used if mounted on vehicle's windows; many applications can be built using the videos or pictures captured by smartphone's camera.

Microphone is a common sensor used by different domains. It has been used in some ITS applications like a roadside-installed microphone picks up the audio that compares the various vehicle noises was used in [76] to measure the traffic density on a road. In the same manner, without the need of installing a roadside microphone, the microphones within smartphones can be used to receive the audio signals generated by tire noise, engine noise, honks, and air turbulence noise.

Another set of uncommon sensors that are rarely used in ITS applications are the **Proximity sensor** and the Light sensor. The proximity sensor is provided to estimate the distance between the face of a device and an object, manufacturers include the proximity sensors in their hand-held devices to determine when a handset

is being held close to a user's face.

The **Light sensor** is used by most devices to control the screen brightness. Light sensors can be used by developers to understand the environment around the driver.

3.2.2 DATA FILTERING AND PREPARATION MODULES

In this module, the collected data will go through multiple steps of fusion, noise reduction, and filtering. Sensors raw data are noisy and meaningless if they are not filtered. Accelerometer data carry biased values and they can get affected easily by small shocks, while gyroscope data are more stable but suffer the drift problem. Magnetometer data get affected by the surrounding environment and they are known to produce inaccurate data indoors. GPS is not reliable indoors, in tunnels or near tall buildings, also in case of good signal reception the error can go up to 15 meters. The framework fused the raw readings from multiple sensors to get more accurate sensors data.

This module applies different types of filtering and processing techniques to the raw data. These techniques will be applied to different type of sensors. For example, detecting and fixing the accelerometer bias and fluctuating problem. The accuracy of accelerometers can be tested by conducting a simple experiment, by placing a smartphone on a flat surface and read the accelerometer data. The readings are expected to be constant, since the phone and the surface are not moving, but the readings will keep fluctuating as if the phone is moving. Each axis in the accelerometer has bias value that needed to be detected and removed. This module will be also responsible for fixing gyroscope drift problem. Gyroscope is important in detecting lane switches, turns and calculating turns' angles. Gyroscope readings suffer from the drift problem, the need to be fixed using the other more stable sensors. One of the most important modules in this layer is the module responsible of making the readings orientation independent. Readings of inertial sensors are referenced to the phone's local frame, where the X axis points to the side of the screen, the Y axis points to the top of the screen, and the Z axis is perpendicular to X and Y. To use the readings of the inertial sensors we need to relate the readings to the vehicle's frame. To do this, the framework translates the data from the phone's local frame to the vehicle's frame.

3.3 VEHICLE DYNAMICS EXTRACTION LAYER

This layer reads sensor readings and extract different features from them. The features extracted in this module are divided into two categories, the basic level features and the advanced level features.

3.3.1 BASIC LEVEL FEATURES

Basic level features are the features that explain vehicle's basic motion dynamics as described in [77], [78], [79], [62].

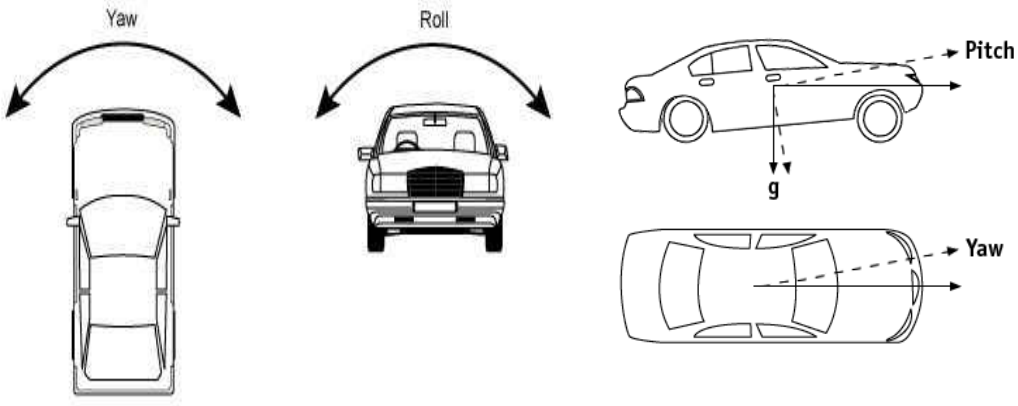


FIG. 9: Basic motion dynamics the car can take.

Works that studied the basic vehicles motion dynamics have all agreed that vehicles have six basic motion dynamics. First the vertical motion which is vertical to the ground, where the force is positive if upward and negative if downward. The vertical motion is also called the vertical force or the vertical load. The second is the lateral motion which is the motion toward the side of the vehicle. It is generated when the vehicle takes a turn or a lane switch. The third motion is the forward motion that is generated when a vehicle is driving in a straight line, either forward or backward. It is positive if the vehicle is moving forward and negative if the vehicle is moving backward. The fourth motion called the yaw rate which is the movement about the z-axis. It can be used to calculate the turning angle. The fifth motion is the pitch

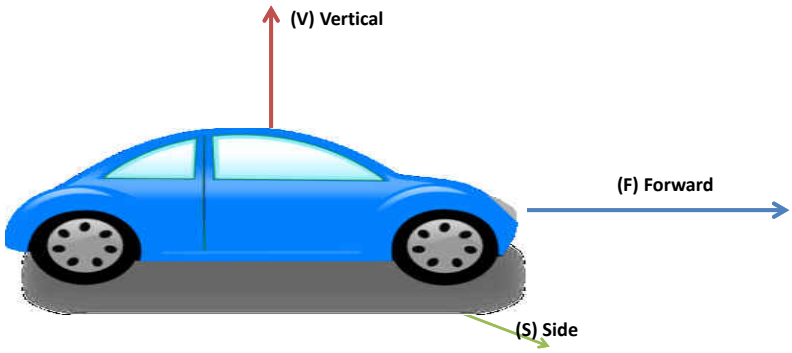


FIG. 10: Vehicle Coordinates.

rate. It is the lateral movement about the y-axis. The sixth and last motion is the roll rate which is the movement around the x-axis.

3.3.2 ADVANCED LEVEL FEATURES

The other category of features is the advanced features which contains five modules. Previous work [37,38,39,41,45] has focused on vehicles dynamics and road traffic, and they are collectively trying to extract different motion features using GPS, road-infrastructure, OBD and other means to collect different dynamics. In this work we find the activities that are related to vehicle's dynamics and included them in the SenSys framework, given that, the framework is expandable and can accept new activities.

The first module in the advance features category is the speed estimation. Vehicle's speed estimation is very important feature that is needed by many applications. Speed can be retrieved directly from OBD or GPS units. This feature become hard to calculate when the applications depend on the inertial sensors only, this feature is needed since more application tend to overcome the unavailability of the OBD devices and the GPS signals by depending on the inertial sensors only. The second module is the turn detection, where turns can be detected using GPS, OBD, and inertial sensors. In addition to unavailability and the accuracy problem, GPS is late and not sensitive to small turns. SenSys can detect turns and angles taken using inertial sensors. Inertial sensors are sensitive and more accurate in detecting turns

and lane switches. The third module is the stop detection module. Stops can be detected by retrieving the speed from GPS and OBD, but experiments showed that GPS and OBD are not very accurate in very low speeds, thus cannot detect the exact time when the stopping event occurred. Some applications need the exact start and end time of the stops to calculate the fuel consumption, queue length, and waiting time in intersections. The lane-switch detecting module uses inertial sensors to detect lane changes. GPS and OBD are not sensitive to detect lane switches. The 10 meters GPS accuracy is not enough to detect the change in vehicles lane. SenSys uses inertial sensors to detect lane switch, it can differentiate between lanes switched and turns. This can be used to enhance navigation systems, guide emergency cars, and identify vehicle's lane. The fifth module is responsible for bumps and potholes detection. This module can provide the developers with information about roads surface, bumps and potholes that are detected by SenSys. It can be used by many applications to understand roads surfaces and to enhance the navigation systems and the safety on the road.

The developers can specify the sensors that can be used to extract the features based on the application requirements. Since some applications might use inertial sensors only while other applications can use other sensors like OBD or GPS. The methods of extracting the features will differ based on the sensors used. For example while the speed estimation is straight forward using the OBD or GPS, it is complicated using the inertial sensors only. Many developers prefer to use the inertial sensors only because of their availability, since GPS is not available in some cases and the OBD needs an extra device which is not available with most drivers.

3.4 APPLICATION PROGRAMMABLE INTERFACE (API)

The API layer provides a set of calls that can be used by developers to access the different features of the framework. This layer acts as an interface between the data processing and features extraction layers and the application layer. Applications use the calls from the API layer to access the filtered sensors data and vehicle's dynamics features. The API calls provided by SenSys can be used by different types of applications, for example: navigation application, road safety applications, applications that analyze road traffic, applications that test the driver behavior, application that test road surface conditions, and others.

This layer can be expanded by adding new API calls from new applications.

3.5 APPLICATIONS

The application layer consists of different types of applications that use the information provided by the API layer. These applications will provide developers with rich set of information that can be used by many ITS applications.

SenSys application layer implemented a set of applications, for example an In-lane communication system which uses the services provided by different layers of SenSys to extract road and traffic features of a lane. This will help a vehicle to identify and communicate with all other vehicles within the same lane. Another application which is part of the SenSys is the ParkZoom which is a parking spot identification system. This system uses the turn detection and speed estimation services provided by SenSys to estimate the exact parking spot. This system helps drivers to find a free parking spot in a large crowded parking lot, it also help them to find their car in the parking lot.

Fuel consumption and CO_2 Emission Calculator(GoGree) uses inertial sensors to calculate the fuel consumption and the CO2 emission. Giving a real-time feedback to drivers about their fuel consumption and CO_2 emission will help them to change their driving behavior to be environment friendly.

Vehcile type esitmator detects the number of vehicle's axles and the distance between its axles to estimate its class. FHWA has divided the classes into 13 class based on the number of axles and the size of the vehicle. All of these applications will be explained in detail in Chapters 7,8,9 and 10.

CHAPTER 4

DATA PREPARATION AND PRE-PROCESSING

4.1 INTRODUCTION

In this chapter we will explain the set of modules used to filter and prepare the raw data in the data preparation layer. First, it will start with filters applied to individual sensors like low pass filter, complementary filter and accelerometer bias filter. Then, it will explain in detail in the coordinate alignment module.

4.2 LOW PASS FILTER

The first filter we apply to the raw data is the low pass filter [80]. It is needed because low cost accelerometers used in smartphones are considered noisy and their readings are fluctuating all the time. If we put the phone on a fixed surface we will get continuous fluctuating readings even if neither the phone nor the table are moving. The algorithm we are designing is sensitive to changes in accelerometer readings. Therefore we used a moving average low pass filter to remove this fluctuating. Figure 11 shows the accelerometer readings in the direction of motion before and after applying the filter. From the figure we can see that all activities like accelerating, decelerating, stops, and others still exist and the readings are smoother and easier to analyze.

4.3 COMPLEMENTARY FILTER

The goal of this filter is to correct the gyroscope drifting problem. The gyroscope has to be used to calculate the change in the angle taken over time, but because of the integration over time, the angle calculated will drift from the original position. Since the gyroscope data is precise and reliable in the short term but tends to drift in the long term, it can be corrected by the accelerometer readings that do not drift over the long term. Correcting the gyro's drift by more reliable source can be done by either complementary filter [81] or Kalman filter [82]. SenSys uses the complementary filter

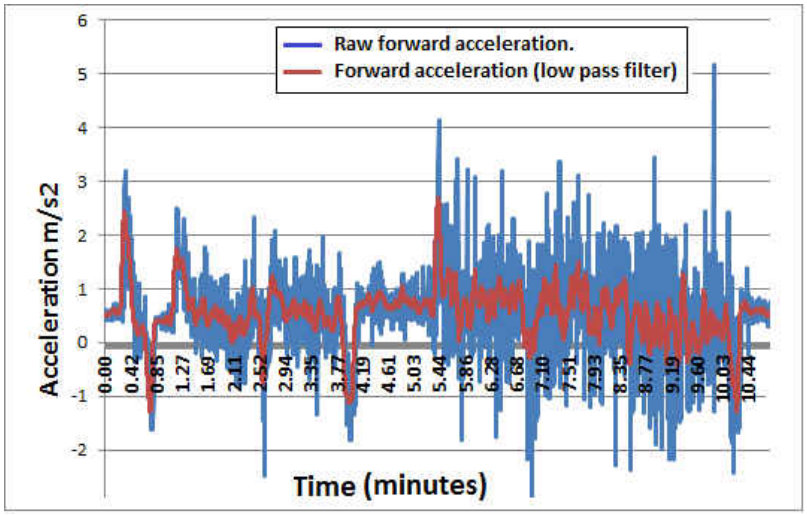


FIG. 11: Using low pass filter (moving average) to reduce the fluctuation

because it is simpler to implement and has the same accuracy as the Kalman filter. Figure 12 shows the complimentary filter diagram.

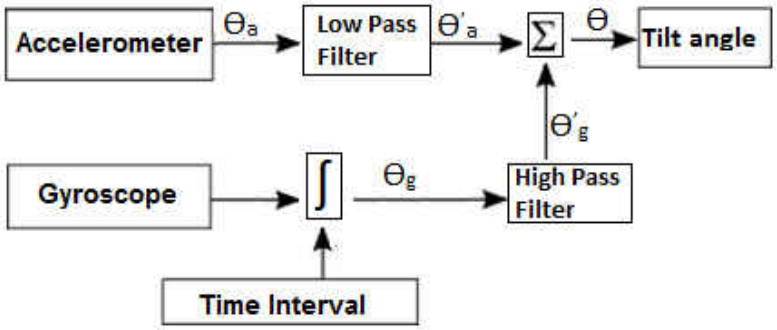


FIG. 12: The complimentary filter diagram

In the diagram $\theta_a = arctan(a_y/a_z)$ and θ_g is the dynamic tilt angle calculated by integrating the angular velocity (gyroscope reading), and $\theta = \theta'_g + \theta'_a$.

Although the gyroscope is more stable and less noisy than the accelerometer in detecting the changes in small intervals, but in the long run, the gyroscope will drift and therefore needs calibration. We built the complementary filter in a similar way to [81]. Figure 13 shows the calculated angle in degrees before and after the drift correction.

4.4 ACCELEROMETER BIAS

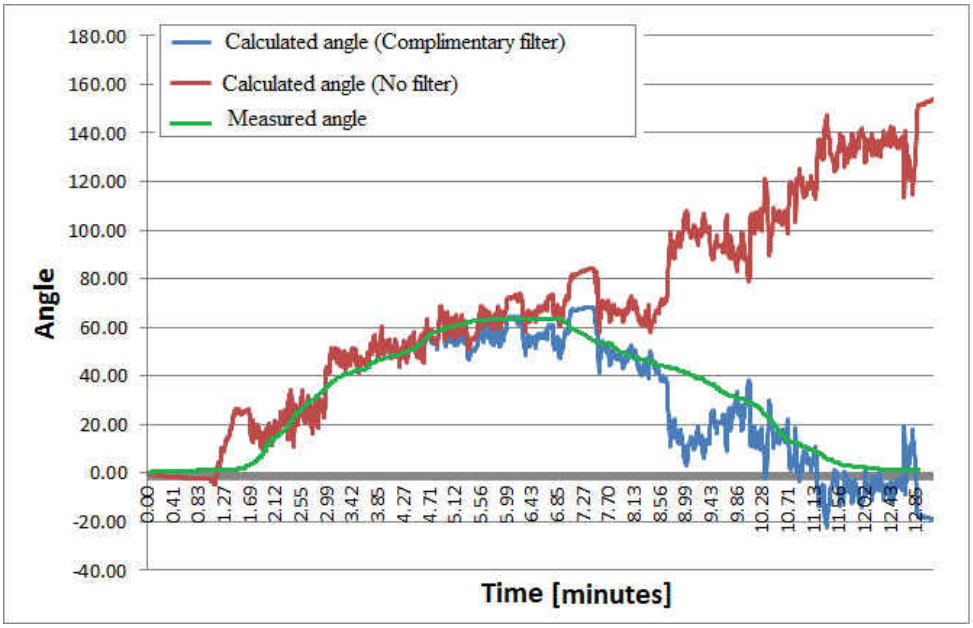


FIG. 13: Calculated angle before and after removing the drift.

In smartphones, accelerometers are very sensitive in a way that they can detect small changes in the acceleration, but these readings could be inaccurate. Accelerometer readings add additional offset to each axis of the phone's coordinates, this value will be referred to as the accelerometer bias. To study this bias and calibrate it, we obtained a simple test to measure the accuracy of the accelerometer in smartphones. We collected the accelerometer readings from a Samsung Galaxy smartphone placed on a flat table and show the results in Figure 14.

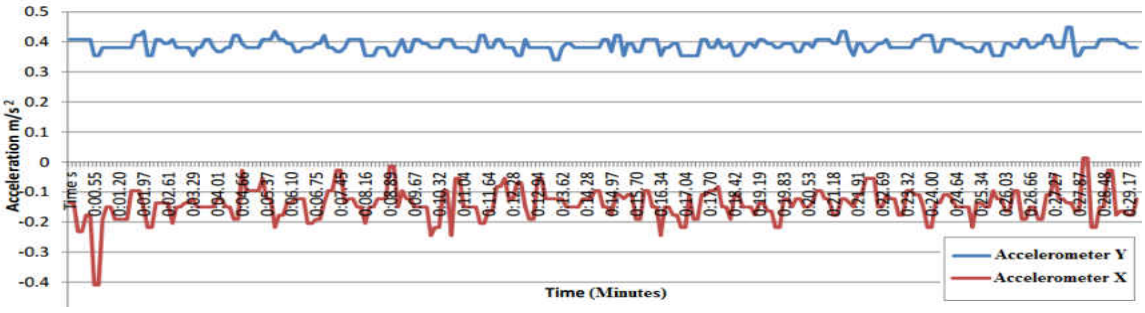


FIG. 14: Accelerometer readings while the phone is placed on flat table.

In Figure 14 the average value of the Y axis readings is 0.39 and the average value

in the X axis is -0.14. Since the phone was stable and placed on a flat surface, the readings are expected to be zero on both axes. The values read by the accelerometer in this situation represent the accelerometer bias. This test showed that the accelerometers readings are biased and each axis has different bias value. In addition to that, when studying the values produced by other phones as shown in Table 5 we found that the bias differs from phone to phone, even for the phones with the same model and operating system, also the bias differs from axis to axis within the same phone.

TABLE 5: Accelerometer bias measured using different phones

Phone	X axis	Y axis	Z axis
Galaxy S4	-0.14	0.39	9.84
Galaxy Note 1(1)	0.79	0.09	9.87
Galaxy Note 1(2)	0.24	-0.07	9.85
Nexus 4	0.44	0.21	9.81

To study if the bias is consistent per phone, or if it changes from stop to stop, we applied another experiment where we placed a Galaxy S4 phone on the windshield, drove a car for 60 minutes and collected the readings during the multiple stops we did through the experiment. Table 6 shows the bias readings for the same phone in different stops. In each axis the bias is almost constant.

TABLE 6: The accelerometer bias for the same phone in different stops

Phone	X axis	Y axis	Z axis
Stop 1	0.245	9.78	0.188
Stop 2	0.239	9.79	0.179
Stop 3	0.232	9.81	0.191
Stop 4	0.244	9.83	0.187

Since the bias is constant per phone, we can calibrate for this bias by measuring such bias for each sensor offline and then apply a compensation to the raw readings of the sensor to counter the bias. Figure 15 shows the speed calculated before the accelerometer bias calibration process and after the calibration process. The figure shows the importance of the bias calibration process in increasing the accuracy of the readings. To calculate the speed using inertial sensors, we integrate the acceleration over driving time. The bias is included in the calculation, so, overtime, this will increase the error in the speed as shown in Figure 15. To remove the bias from the

accelerometer readings, we calculate it offline where we position the phone in different orientations where readings along axes are expected to be zero. After removing the bias from the accelerometer readings, the comparison will be more reasonable as seen in Figure 15. The calibrated speed is calculated using the following formula:

$$Speed_{i+1} = Speed_i + (A_i - bias) * dt \tag{1}$$

where A is the acceleration, dt is 0.1 second, and the sampling rate is 10 readings per second. Our previous work [18] explains in detail how to detect stops using inertial sensors only.

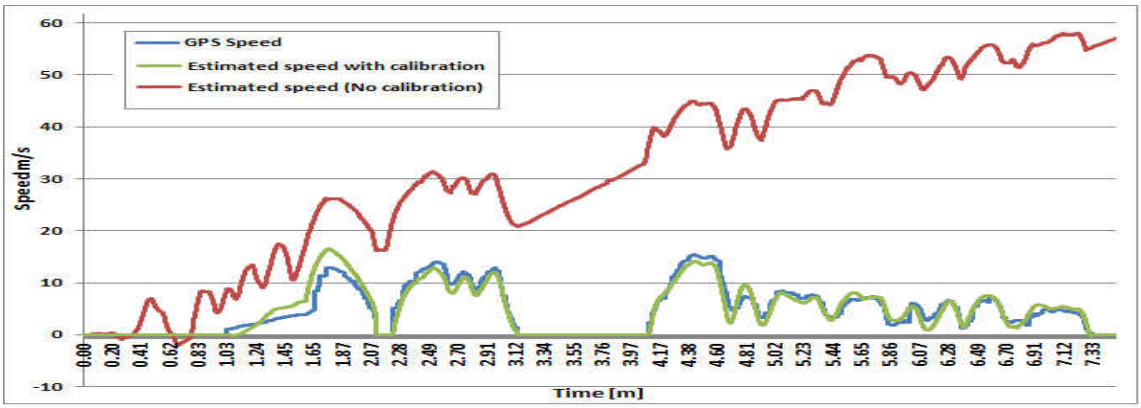


FIG. 15: GPS speed vs estimated speed before bias reduction using our algorithm, vs using raw speed acceleration from the phone Z coordinate

4.5 COORDINATE ALIGNMENT (UNICOOR FRAMEWORK)

Translating sensor readings from phone to vehicle context is one of the challenges faced in using smartphones inertial sensors for ITS applications. Existing solutions for this challenge are divided into three approaches, the first approach used in [19, 36, 55] avoids the problem by fixing the phone on a car mount and sets its orientation to be aligned with the vehicle coordinate system. Adapting this approach limits the driver from using his phone while driving; also, it could produce inaccurate readings if the phone's orientation tilts because of a hard break or a road bump. The second approach used in [56] is an application specific solution. The methods used in the second approach work only with specific applications and specific scenarios and cannot be used in other applications. The third approach used in [57] utilizes the GPS to recognize the vehicle's direction of motion.

Although GPS readings could be used to estimate vehicle's direction of motion, GPS readings suffer from several limitations. One major limitation is the high power consumption associated with GPS readings. Moreover, GPS readings are not available or have very high inaccuracy in tunnels, indoor parking-lots, and urban areas with tall buildings and trees. Figure 16 shows the power consumed in a trip. In the first half, the first application used the inertial sensors only to estimate the motion direction, and in the second half the application used the GPS only to estimate the motion direction. The experiment shows that the GPS average power consumption is 1917 mW and the power consumption using the inertial sensors only is 1321 mW, which is about 30% reduction in power consumption. Therefore, our approach in this work is to utilize phone inertial sensors only without any help from the GPS or the OBD.

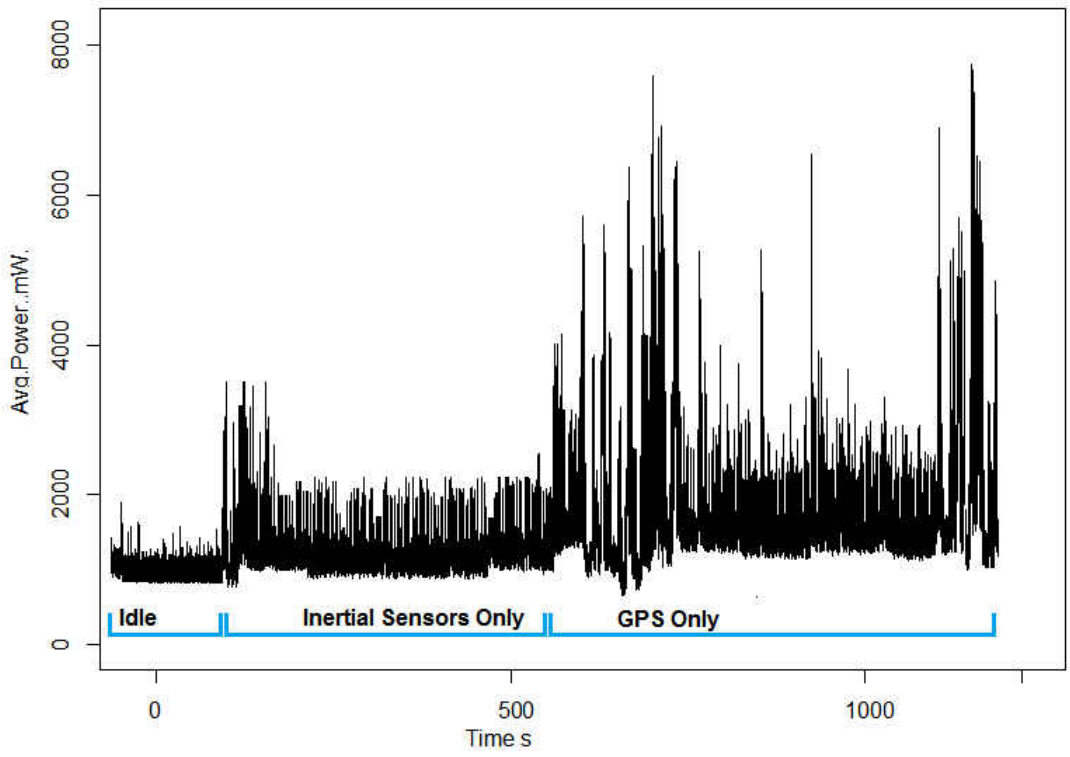


FIG. 16: Power Consumption in mW

In this section we explain, develop, and evaluate a reliable framework (UniCoor) that uses inertial sensors only to map the inertial sensors readings from the device coordinates to the vehicle coordinates. This framework will enhance the accuracy of

tracking and analyzing various vehicle dynamics such as vehicle stops, lane changes and accurate vehicle speed calculation that, in turn, will enable development of new ITS applications and services. As a proof of concept of our framework, we use our framework in estimating a vehicle's speed using the forward acceleration. We use the GPS speed as the ground truth to evaluate our framework accuracy in estimating vehicle speed.

4.5.1 PHONE VERSUS VEHICLE COORDINATES

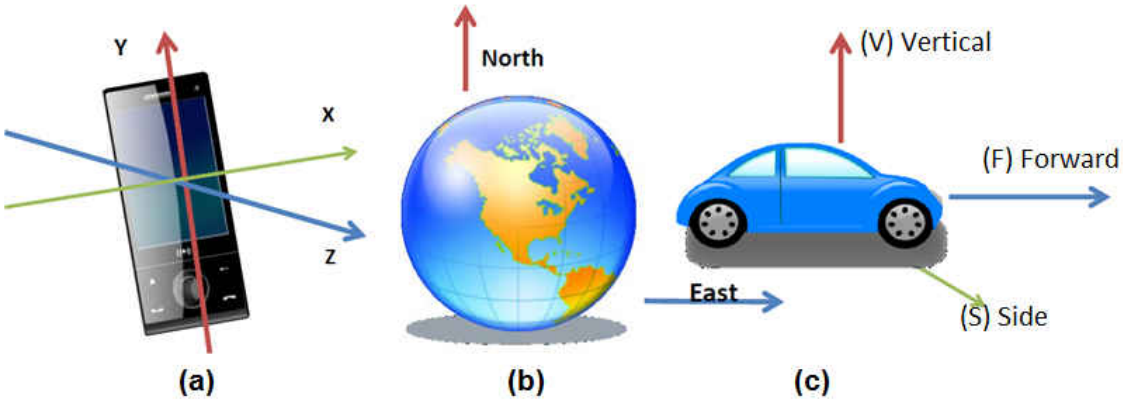


FIG. 17: Coordinate systems. (a) Device Coordinate System, (b) Earth Coordinate System, (c) Vehicle Coordinate System.

Figure 17 shows the three types of coordinate systems used in the framework:
Device Frame Coordinate (DFC) is relative to the screen of the smartphone. As shown in Figure 17(a) the X axis points to the right of the screen, Y axis points toward the top of the screen, and Z axis is perpendicular to X and Y, it goes through the screen and points toward the front face of the screen. Inertial sensor readings are in reference to the device coordinates. Since the device's axes are relative to the phone's frame, the X axis will keep pointing toward the side of the screen even when the orientation changes.
Earth Frame Coordinate (EFC) corresponds to the Geographic Coordinate System. It consists of the Gravity axis (G) that points toward the center of the earth, the North axis (N) that points toward the magnetic north, and the East axis (E) which points toward the east direction and is perpendicular to Gravity and North,

as shown in Figure 17(b).

Vehicle Frame Coordinate (VFC) is shown in Figure 17(c), as is composed of the Forward axis (F) which points toward the direction of motion, Side axis (S), and the Vertical axis (V) which is perpendicular to F and S and points downward. Inertial sensor readings in VFC reference are aligned with vehicle movement. Therefore, these readings could be utilized in extracting accurate vehicle dynamics.

Mobile smartphones come with inertial (motion) sensors, typically, 3D-accelerometer, 3D-magnetometer and 3D-gyroscope. All inertial sensors have sense of the three dimension space, therefore each sensor will produce three readings and each reading is in reference to one of the device's axes. For example, the acceleration sensor produces 3 reading values in X, Y, and Z axes. The acceleration in X axis is corresponding to the phone acceleration along the device X axis in m/s^2, which represents the acceleration toward the side of the phone. Another example is X reading of the gyroscope sensor, which is corresponding to the rotation velocity of the device around the X axis in radians per second. Sensors reading values are relative to phone's coordinate; therefore, a simple change in phone orientation while it is in a moving vehicle would affect significantly the inertial sensor readings.

4.5.2 IMPACT OF COORDINATES

Aligning device and vehicle coordinate systems has a great impact on sensor readings. The raw readings detect the motion of the phone in reference to its local axes. If a vehicle is moving and its coordinates are not aligned with the phones' coordinates, then all readings will recognize the device's motion but not the vehicle's motion.

To show the coordinates' impact, we conducted an experiment with two phones; the first one was placed in an arbitrary orientation, while the second phone was aligned with the vehicle coordinate (i.e., DFC is aligned with VFC). Figure 18 shows that the Z axis of the second phone is aligned with the vehicle's F (Forward) axis. Accelerometer readings were used to calculate the speed of the car in both phones. GPS speed is collected to be used as ground truth. Figure 19 shows how the speed calculated using the aligned phone matches the speed collected by the GPS, while Figure 20 shows the speed calculated by the tilted phone does not match the GPS speed. The results of this experiment shows the importance of aligning device's coordinate with vehicle's coordinate and how sensors' readings in aligned phones can

FIG. 18: Phones are fixed on the windshield with different angles

be utilized to detect vehicle's dynamics like speed calculation, detecting stops, and turns. The graph also shows the speed calculated using the three axes X,Y, and Z after removing the gravity.

4.5.3 SYSTEM DESIGN

In this section, we describe our framework and its different components.

UniCoor Framework

UniCoor (Unified Coordinate System) is a framework that aims to transform the sensor readings to a unified coordinate system that would be useful in transportation. The most appropriate coordinate system in transportation is the vehicle coordinate system and hence the goal of UniCoor framework is to transform the readings from the phone coordinate to vehicle coordinate. In the UniCoor framework, sensors data goes through multiple modules to be mapped to vehicle coordinate as shown in Figure 21. Sensor raw readings go to the data filtering module which filters noisy sensors readings and prepares them for processing in the next modules. Filtered

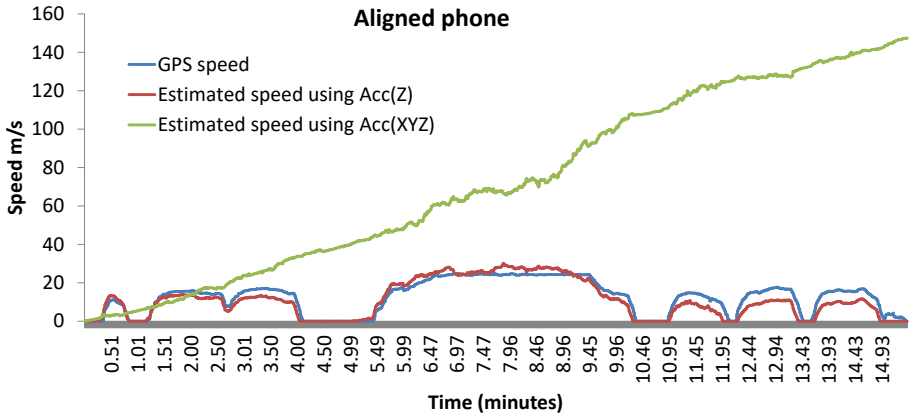


FIG. 19: Calculated speed using aligned phone

data will be sent to the next module (DFC to EFC) which maps the readings from device frame coordinate to the Earth frame coordinate. The DFC-to-EFC module filters out the gravity vector and sends a 2D vector that represents the readings in the horizontal plane. The next module consists of three submodules; the first submodule takes the 2D vector and divides it into small windows. The second sub-module (Angle estimation) applies the principle component analysis (PCA) on each window separately to find the angle between the earth coordinate and the coordinate with the highest variance in acceleration. And the third sub-module is the accuracy enhancement sub-module which takes the estimated angles and analyzes them to filter out the inaccurate estimations. The enhanced estimated angles will be used in the (DFC-to-VFC) module to map the data from the Earth coordinate to the vehicle coordinate. The output of this framework will be the sensors readings in reference to vehicle coordinate system that can be used by other developers to create ITS applications

4.5.4 COORDINATES ALIGNMENT

The filtered data will be sent to the next modules to align the readings with the desired orientation. This section explains the three modules that transform the readings from DFC to VFC. First, readings get mapped to EFC to find the gravity coordinate and get the readings in horizontal 2D-plane without the gravity effect. The data in 2D horizontal coordinates will be sent to the PCA module which finds

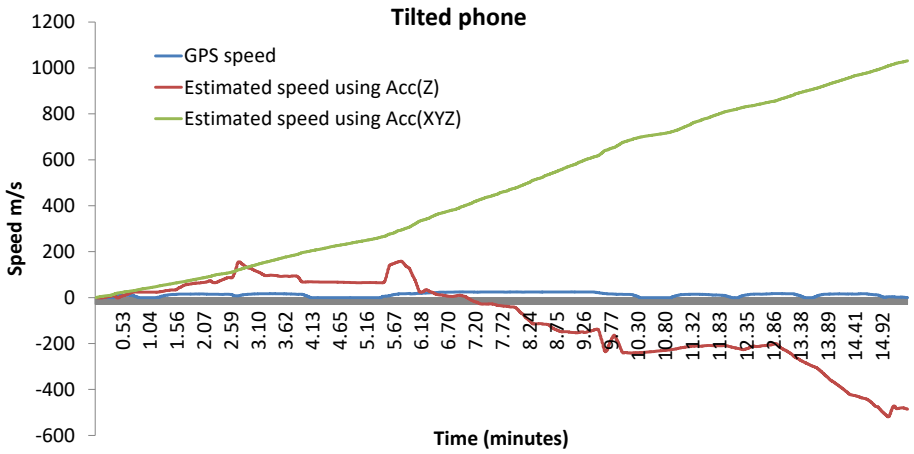


FIG. 20: Calculated speed using the tilted phone.

the direction of motion that would be used to detect the vehicle coordinates.

Device Coordinate to Earth Coordinate (DFC to EFC)

This module maps accelerometer readings from device coordinate to Earth coordinate. It uses the accelerometer, gyroscope, and magnetometer sensors to detect the gravity coordinate that represents the vertical motion. It assumes the other two dimensions represent the horizontal motion of the car. The horizontal motion is the motion in the forward and side directions as in Figure 17.

This module and the next modules depend mostly on the accelerometer readings, and the following sub-modules will explain the steps taken to map the coordinate from DFC to VFC.

The accelerometer readings in DFC and EFC will be represented in the following matrices:

$$A_{DFC} = \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix}, A_{EFC} = \begin{bmatrix} A_N \\ A_E \\ A_G \end{bmatrix} \tag{2}$$

where A_x is the acceleration along the X axis, and X, Y, and Z are the device's axes. N, E, and G are the Earth coordinates North, East and Gravity.

To transform the readings from device coordinate to Earth coordinate, we should find the rotation matrix needed to map the DFC to EFC. To do that, multiply the

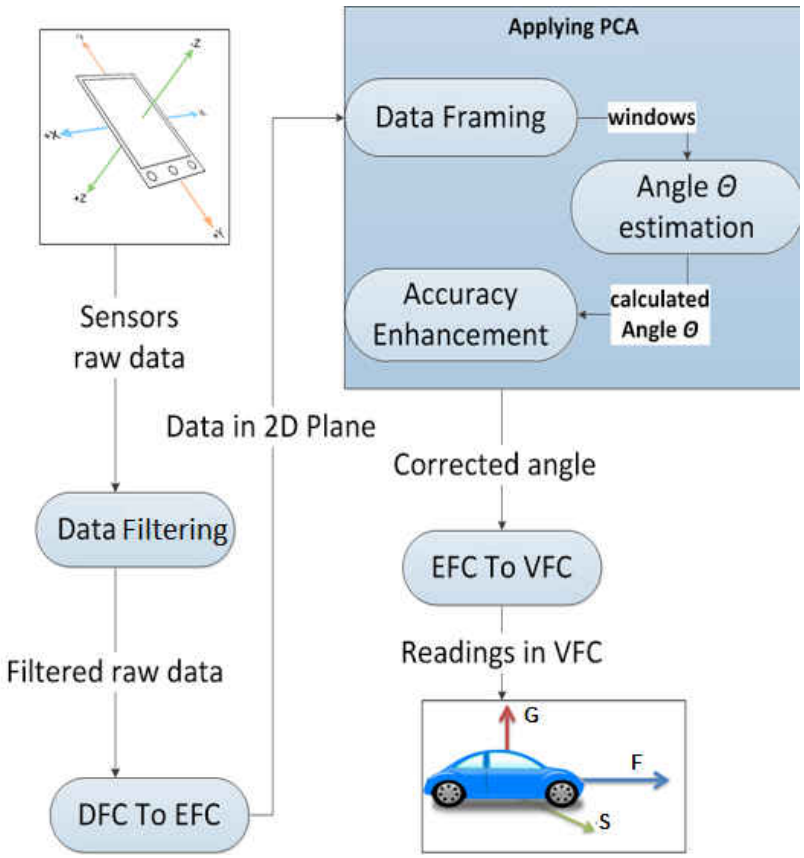


FIG. 21: System framework. DFC: Device Frame Coordinate, EFC: Earth Frame Coodrinate, VFC: Vehicle Frame Coordinate, theta : angle between the direction of motion and the magnatic north, 2D: horizontal plane.

DFC readings by the rotation matrix using the following equation:

$$\begin{bmatrix} A_N \\ A_E \\ A_G \end{bmatrix} = \begin{bmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{bmatrix} * \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} \tag{3}$$

where R is the rotation matrix returned by *getRotationMatrix*, A_E, A_N are the horizontal axes, and A_G is the gravity axis.

The DFC to EFC module uses set of functions provided by the Android API. Android provides set of functions that can be used to determine the phone orientation and to detect the changes in the orientation. These functions are mostly used to adjust the view based on the screen orientation. The system uses the Android function *getRotationMatrix* that takes the readings from the accelerometer and the

magnetometer as parameters. This function produces the rotation matrix which will be used to transform the raw readings of an inertial sensor (e.g., accelerometer) from the device coordinate reference to the earth coordinate reference.

The output of this module will be sent to the next module that applies the PCA to find the vehicle's direction of motion.

Applying PCA

The PCA module takes the horizontal 2D-plane of the Earth coordinate of the accelerometer readings and applies on them the principle component analysis (PCA) to find the direction that has the highest variance in acceleration. This direction will be aligned with the forward motion direction of the vehicle. The goal of this module is to detect the angle between the direction of motion and the EFC. This angle would be used to map the readings from horizontal earth frame coordinate (EFC) to to vehicle frame coordinate (VFC).

This module is divided into three sub-modules:

- The data framing sub-module which divides the input stream into windows.
- Angle estimation sub-module which applies the PCA and calculates the angle between the two coordinates in the current window.
- The accuracy enhancement sub-module which analyzes and corrects the calculated angle because each window will have different accuracy. The techniques used in this sub-module explained in details in the evaluation section.

Data framing: Before applying the PCA on the data, the system divides the input stream into windows, each window consists of 20 readings. It is important to calculate the orientation angle more frequently to detect any change in the orientation in a short time. In the other hand, the PCA needs more time to collect enough data to calculate the new angle. To solve this contradiction, the algorithm splits the data into windows; each window will represent part of the readings, which is almost two to three seconds per window. In each window, the algorithm reads the two vectors that represent the accelerometer readings in the horizontal plane after removing the gravity vector. For each window the PCA module will produce the

angle between the calculated coordinate and the Earth coordinate.

Angle Estimation: The angle estimation algorithm is applied to each window separately, the resulting angles of each window would be sent to the next module for processing and filtering. The angle between two 2D horizontal coordinates is needed to transform the readings from one coordinate to another. The EFC coordinate is known, but the vehicle coordinate is not known up to this point. The VFC coordinate consist of the forward axis, side axis, and vertical axis. The vertical axis is known because it matches the gravity axis in the EFC. Detecting one of the remaining coordinates will give the other one because the angle between them is always 90 degrees on the horizontal plane. PCA uses the high variance caused by the acceleration and deceleration to detect the direction of motion and therefore the VFC.

PCA calculation steps:

- For each window
  - Calculate the co-variance matrix:

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X - X^{-})(Y - Y^{-})}{(n-1)} \tag{4}$$

$$C = \begin{bmatrix} cov(x,x) & cov(x,y) \\ cov(y,x) & cov(y,y) \end{bmatrix} \tag{5}$$

  - Calculate the eigenvectors of the covariance matrix.
  - Choose the eigenvector that correspond to the highest eigenvalue. If A is a square matrix, B is an eigenvector of A if there is an eigenvalue v such that: AB=vB

The output of the PCA procedure will be the angle between the eigenvector coordinate and the EFC, this angle could be used to remap from one coordinate to another. The next step is to send the results to the accuracy enhancement sub module.

Accuracy Enhancement: The accuracy enhancement sub-module takes the estimated angles calculated by the PCA to enhance the accuracy of the estimation. It filters the data by analyzing the relation between sensors readings and the resulted

angles. This step is needed because of the high variance of the calculated angles in each window, this variance happens even if the vehicle is driving in a straight line. The accuracy enhancement sub-module is explained in detail in the evaluation section.

Earth Coordinate to Vehicle Coordinate (EFC to VFC)

The final step is to map the readings from EFC to VFC. This step is straightforward given the angles between the coordinates. Figure 22(a) shows the the mapping from DFC to EFC, in this step the earth's gravity axis will be matching the vehicle's vertical axis. The vertical axis is not needed to detect the horizontal axes. In Figure 22(b) the PCA estimates the angle between vehicle's horizontal coordinates and Earth's horizontal coordinates.

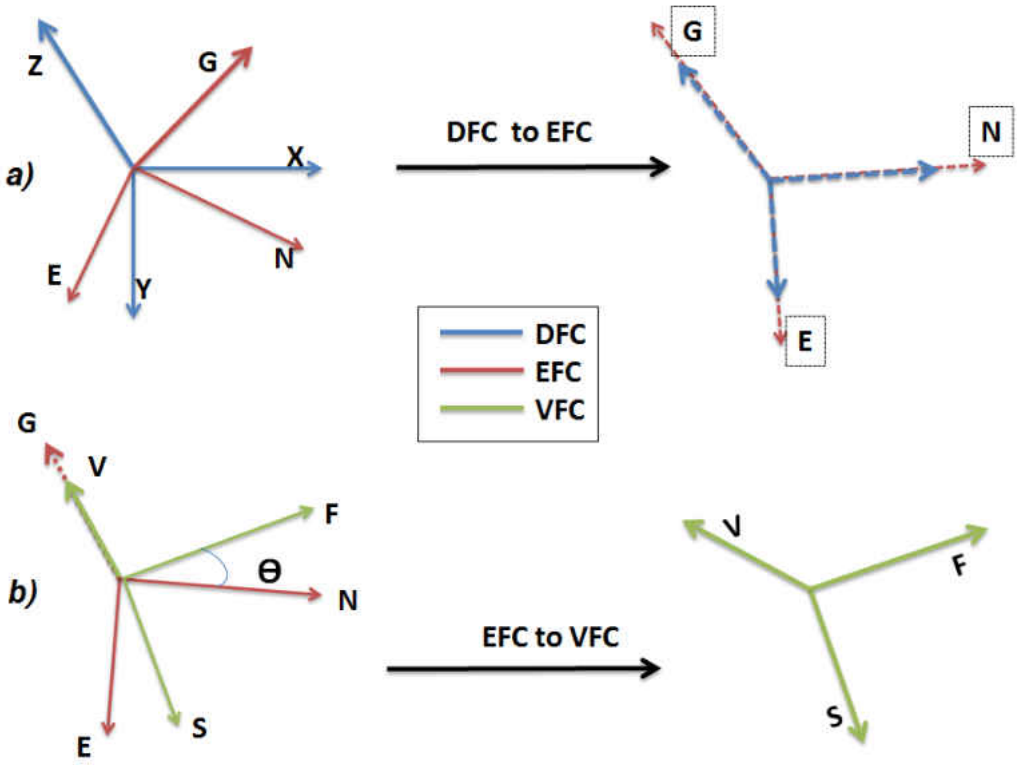


FIG. 22: a) Transform sensors reading values from DFC to EFC. b) Transform readings from EFC to VFC.

4.5.5 EVALUATION

The approach used to evaluate the accuracy of this framework is using the acceleration in the direction of motion to calculate the speed and then compare it to the actual speed collected by the GPS. Getting the correct speed means that the mapping from the arbitrary orientation to the vehicle orientation is successful.

Basic Experiments

Experiment Setup

To test the framework, we developed an Android application that collects, filters, and processes the raw readings from inertial sensors with a 10Hz rate. In addition, it collects the GPS readings to be used as ground truth values during the evaluation process. The phones used in the experiments are Samsung Galaxy S4 with Android 4.4, Samsung Galaxy Note 1 with Android 4.2, and Samsung Galaxy S with Android 4.2. The cars used are Toyota Camry 2001, Honda Accord 2006 and Honda CR-V 2003. The phones were placed in arbitrary orientations in the cars. The cars driven for more than 30 miles and covered different types of roads as seen in the map shown in Figure 23.

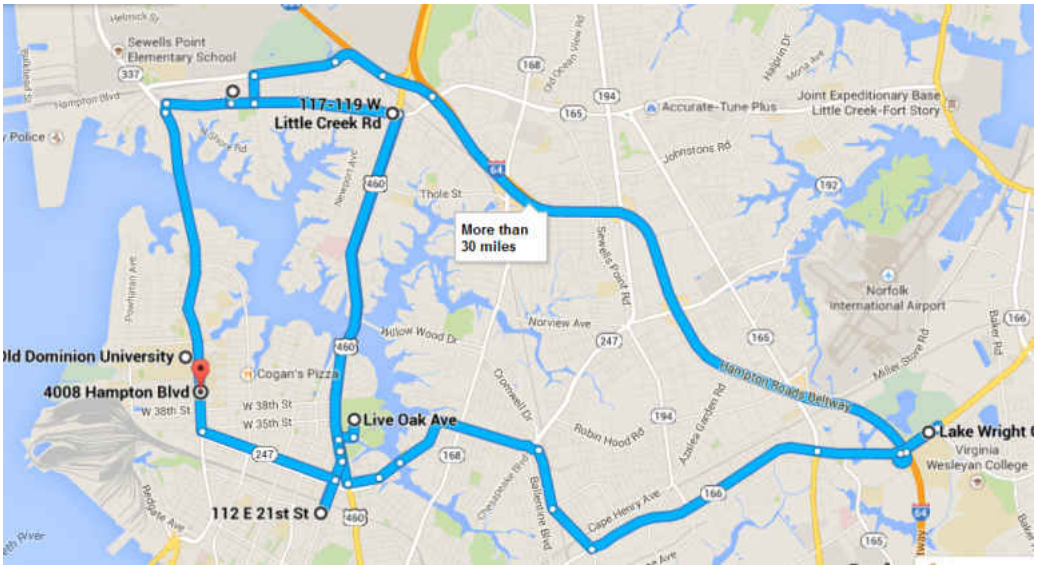


FIG. 23: Roads driven in the experiments

Results

To evaluate the framework, the acceleration in the direction of motion is used to calculate the speed. The calculated speed is compared with the collected GPS speed. After putting the phone in an arbitrary orientation in the car, the algorithm has been used to find the acceleration in the direction of motion and used it to calculate the speed. Figure 24 shows that the estimated speed is matching the GPS speed with 92.7% accuracy, which proves the good accuracy of the algorithm. This accuracy can be enhanced using the accuracy enhancement module.

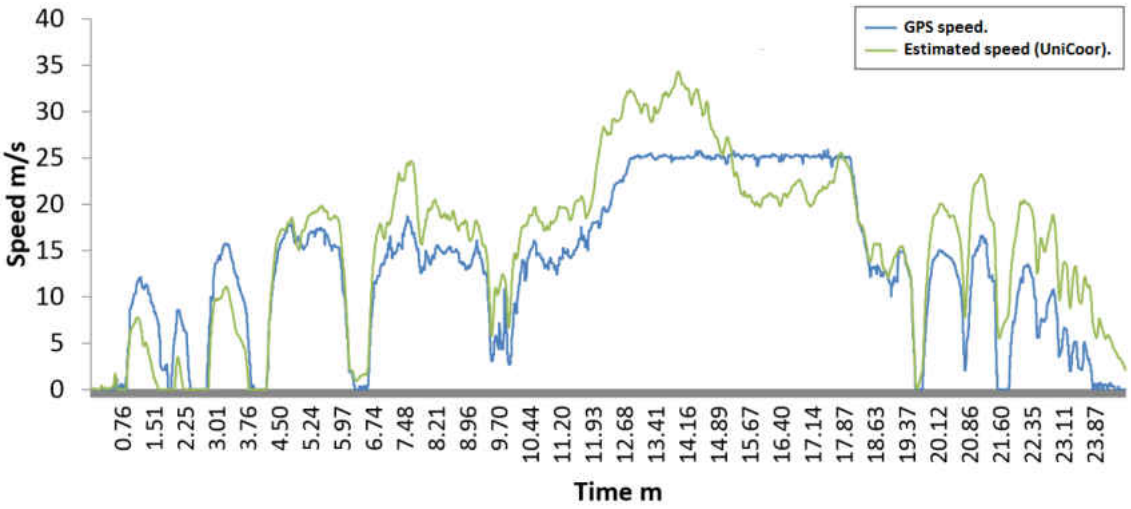


FIG. 24: GPS speed vs estimated UniCoor speed.

Figures 26 and 28 show the results of other trips. The matching between the UniCoor speed and GPS speed proves the successful mapping of the readings from DFC to VFC.

The accuracy of the algorithm depends on many factors, therefore, we explored different methods to test and enhance the accuracy of the framework.

4.5.6 ACCURACY ENHANCEMENT

The accuracy enhancement sub-module is responsible for studying the relation between the acceleration patterns and angle estimation accuracy to enhance the overall accuracy. Experiments showed that each window produces a different angle between the two coordinates (EFC and VFC). To enhance the accuracy of the angle estimation, the system analyzes the relation between the produced angles. Calculating the correct angle between the vehicle coordinate and the Earth coordinate can

be affected by many factors like the noise of the sensors which can affect the DFC to EFC transformation. In addition, the motion status of the vehicle can greatly affect the PCA estimation since it bases its calculation on the variance of the acceleration on the direction of motion. For this reason, different windows will have different accuracy in predicting the correct angle between the coordinates, and the goal of the accuracy enhancement sub-module is to decide if the window has a good or bad accuracy, and based on that, the system can enhance or skip the windows with bad accuracy. Experiments showed that PCA works fine when there is a high and frequent change in the acceleration in the direction of motion like speeding up or down. PCA works fine when the highest variance in the acceleration is in the direction of

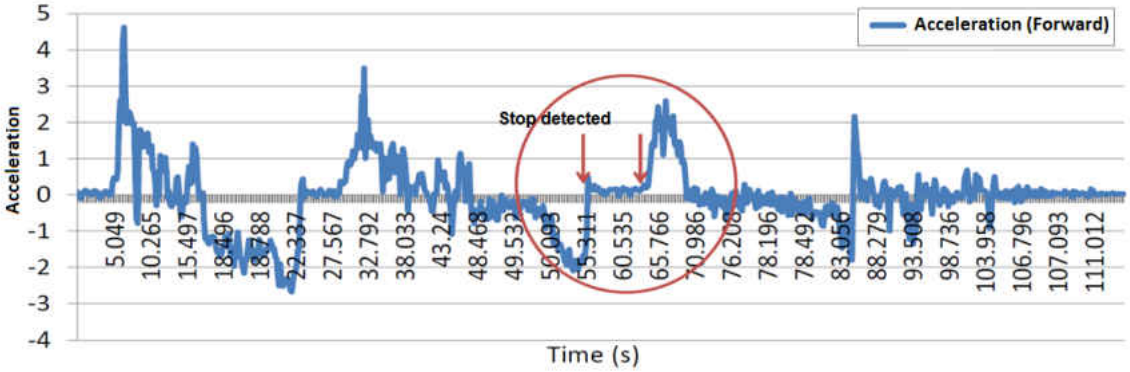


FIG. 25: Accelerations on the motion direction showing stopping patterns

motion, the ideal case is when a car that drives in a flat straight line goes through multiple changes in speed, because deceleration and acceleration patterns produce a high variance. Figure 25 shows the stopping patterns and Figure 26 shows the variance before, during, and after the stopping activity.

The relation between the translation accuracy and the acceleration of the vehicle: First, we show the relation between the accuracy produced and the acceleration of the vehicle. Figure 26 shows the speed collected by GPS and the speed calculated using the acceleration in the direction of motion.

The upper section of Figure 26 shows the GPS collected speed compared with the estimated UniCoor speed. The PCA speed shows estimation accuracy of 92.2% corresponding to the GPS speed. The green line shows the windows where the PCA generates good accuracy estimations. The figure clearly shows the relation between the changes in the speed and the good accuracy where the green line marks the windows before and after the stops. The lower section of the figure shows the

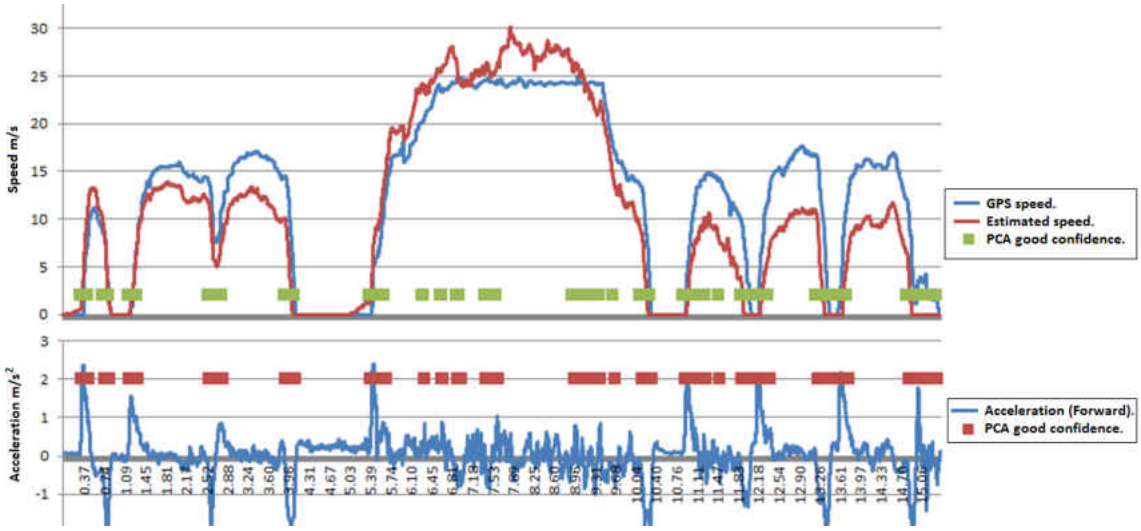


FIG. 26: Good confidence in the accuracy during high changes on the acceleration in the motion direction

acceleration in the motion's direction of the same trip along with the lines that mark windows with good accuracy. The figure shows a tight relation between the change in acceleration and the PCA estimation accuracy. The accuracy is high during high changes in acceleration and low during stops and near constant acceleration.

The relation between the translation accuracy and the variance of the calculated angles: Since acceleration in the speed of direction is not known yet, in this case it cannot be used to filter the data produced by the PCA. So we studied the variance in the calculated angles. Looking at Figure 27, we found that the estimated angle between Earth and vehicle coordinates in each window is fluctuating but when there is a high acceleration in the direction of motion the fluctuating is reduced.

Figure 27 shows that the estimated angles variance is lower than 10 degrees when there is a high change in the acceleration. And it goes to above 30 degrees in case of near-constant acceleration. From this figure, we conclude that, the estimation is expected to have a good accuracy if the variance for the estimated angles is less than specific threshold. Table 7 shows the accumulated error after applying different thresholds on the data, and then calculated the accumulated error between the GPS speed and the calculated speed.

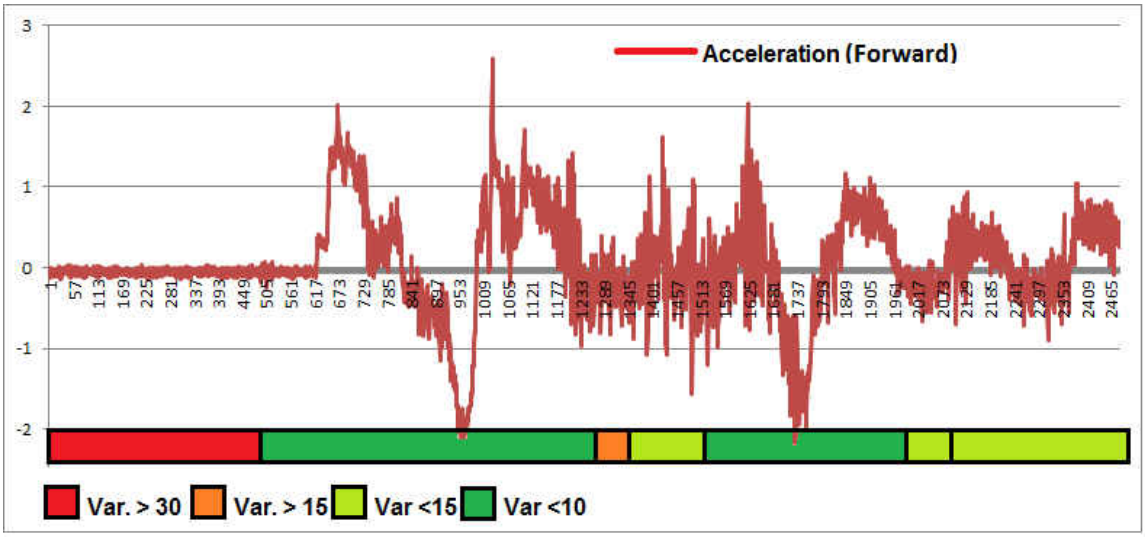


FIG. 27: Predicted angle variance vs GPS speed.

TABLE 7: Accumulated error between GPS speed and estimated speed using different thresholds.

Threshold	No threshold	Threshold 5°	Threshold 10°	Threshold 15°	Threshold 30°
Trip 1	8221	10023	7813	7421	8047
Trip 2	15854	13087	12479	10322	11698
Trip 3	9862	9781	8567	8752	10172
Trip 4	16214	19965	14158	12779	14863

Table 7 shows the minimum accumulated error calculated after using the threshold angle of 15 degrees. Applying a very small threshold like 5 degrees makes the error surprisingly high because it skips lots of important data. A threshold of 30 degrees generates a higher error because it includes more data with low accuracy that could be affected by turns, bumps or any other noise. Therefore, the algorithm uses the calculated angle when the confidence in the algorithm is high. The table shows that the confidence is high when the variance of the estimated angles is less than the 15 degrees threshold if no turns are detected by the gyroscope readings.

The effect of applying the enhancement methods on coordinates translation: Figure 28 shows the results of speed estimation using UniCoor framework before and after the enhancement process. The figure shows that the accuracy of

the average estimated speed after the enhancement is 97.3% corresponding to the ground truth speed from GPS and 85.1% before the enhancement.

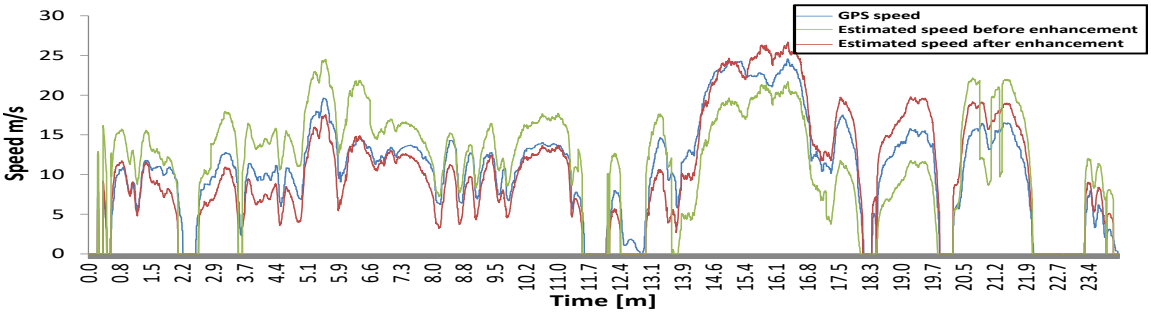


FIG. 28: Speed estimation before and after the enhancement module

Creating a unified coordinate system is a complicated problem because of the nature of the phone sensors and their noisy readings. Smartphones have different types of sensors and manufacturers, and typically sensors used in the phones are small and low-cost, therefore it is expected to find noisy data, and thus the accuracy of the reading values change from one phone to another.

Fixing near constant acceleration: The phone's orientation mapping algorithm depends on the acceleration and deceleration in the motion's direction, and as a result, the accuracy of calculating the right orientation in near-constant acceleration is very low. That is why the algorithm recognizes the intervals when the acceleration is near-constant and uses the angles calculated in previous intervals that have a high variance in the acceleration values. During near-constant acceleration intervals, the variance of the acceleration is very low, so these intervals could be detected and replaced with the angle estimated in the previous interval.

Correct the readings in turns and curves: The acceleration during curves and turns is distributed in the 2D-horizontal direction but not in the forward direction, this will confuse the PCA because this side acceleration might be taken as the motion's direction especially if the change in speed is small. In this case, the system uses the gyroscope to detect turns and then uses the orientation estimation calculated in the last trusted interval.

Overall, in more than 30 miles of driving, the average estimated speed was 96.4% accurate corresponding to the ground truth speed from GPS. The accuracy dropped to 86.3% when we use the algorithm without the enhancement and to 9.7% if the algorithm was not used at all and calculated the speed using the raw readings.

4.5.7 CONCLUSION

UniCoor is a framework that transforms the sensor readings from device local coordinate to vehicle coordinate. The output of this framework is useful in transportation since it enables the inertial sensors to recognize the vehicle's dynamics. In this framework, we use smartphone's inertial sensors only, without the use of the GPS in finding the direction of motion because the GPS is not available all the time, in addition to its high battery consumption. UniCoor framework applies the PCA that uses the changes in the acceleration and deceleration of the vehicle to detect the direction of motion. The evaluation of the system showed a very good accuracy, especially while driving in straight lines. UniCoor is part of the SenSys framework and it can be used by developers to develop ITS applications that use smartphone's inertial sensors to recognize vehicle dynamics.

CHAPTER 5

EXTRACTING VEHICLES DYNAMICS

5.1 INTRODUCTION

This chapter explains the modules in the Vehicle Dynamics Extraction layer. Vehicle dynamics can be divided into two main categories. The first category is the set of basic motion dynamics of the vehicle [77, 78, 79, 62]. The second category is the advanced features and motion dynamics that can be extracted given the set of input provided by the data preparing module. The basic vehicle dynamics include movement or force toward each axis of the vehicle axes and the rotation or the movement about each axis of the vehicle's axes.

5.2 BASIC VEHICLE DYNAMICS

This section explains the basic motion dynamics for a vehicle. There are six vehicle dynamics needed to understand the vehicle movement. These movements are referenced to the vehicle axes showed in Figure 29.

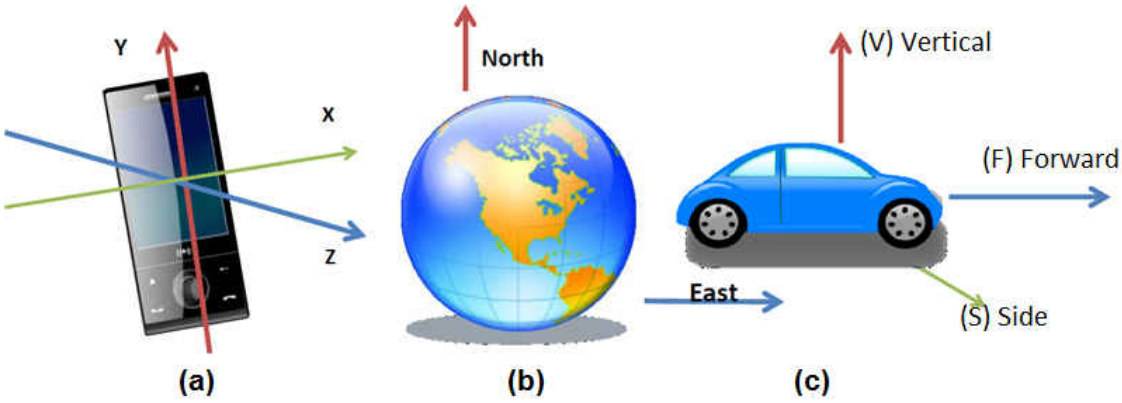


FIG. 29: Coordinate systems. (a) Device Coordinate System, (b) Earth Coordinate System, (c) Vehicle Coordinate System.

5.2.1 FORWARD MOTION

The forward motion is the motion generated when a vehicle drives in a straight line, either forward or backward. It is positive if the vehicle is moving forward and negative if the vehicle is moving backward. According to Figure 29 the forward motion is the motion along the X axis. It can be used to calculate the speed and distance in the forward direction.

$f\_speed_i = f\_speed_{i-1} + Accelerometer(forward) * dt$

$distance_i = distance_{i-1} + f\_speed_i * dt$

where f_speed is the forward speed.

This motion can be used to calculate the speed of the car and the distance traveled. GPS and OBD can get the speed of the car and calculate the distance traveled using the detected speed. But OBD and GPS are not available all the time, and they cannot detect the sudden force applied to the car along its X axis. Inertial sensors are available in smartphones all the time, and they can measure the small shakes and forces applied on moving or stopped vehicles. Using inertial sensor, the forward force can be used to calculate speed and distance of the car. In addition to that it can be used in studying the driving behavior, detecting accidents, detecting stops, and many other applications that can use the acceleration of the car along the X axis (forward axis).

Forward motion can be detected using the accelerometer data in the forward direction, given the phone data is aligned with the vehicle coordinates using the modules in Chapter 4. Figure 30 shows a phone's accelerometer readings of the forward axis recorded while driving on a small trip. The graph shows how these readings can tell when a car accelerates, decelerates, stops, or is driving at a constant speed.

5.2.2 VERTICAL MOTION

The vertical motion is the motion that is vertical to the ground. The force is positive if upward and negative if downward. According to Figure 29 the vertical motion is the force applied along the Z axis (vertical axis). The car moves vertically if it is climbing a bridge, hill, or a road bump. GPS can detect the change in altitude but the average error is 15 meters [83]. GPS cannot detect small changes in vertical motion like bumps, small bridges, or ramps and the OBD does not calculate the vertical motion of the vehicle. Inertial sensors are the best choice to calculate the vertical force applied to the Y axis. The vertical motion used to calculate the slope of

the road and to detect road bumps and potholes. To calculate the distance traveled vertically we use the following equations:

$v\_speed_i = v\_speed_{i-1} + Accelerometer(vertical) * dt$

$distance_i = distance_{i-1} + v\_speed_i * dt$

where v_speed is the vertical speed.

5.2.3 LATERAL MOTION

The lateral motion is the motion toward the side of the vehicle. It is generated when the vehicle takes a turn or a lane switch. This motion is caused by the force applied to the Y axis according Figure 29 and it can be called the side axis. GPS can tell if the car is taking a turn, also OBD can use the data coming from the wheel steering to calculate the taken angle. But both GPS and OBD do not measure the small side movements like lane switch or a shake caused by a bump or pothole. Inertial sensors can use the accelerometer to calculate the acceleration of the car along the side axis. This acceleration can be used to calculate the distance or the displacement of the car toward one of its sides. The side displacement can be used to accurately detect lane switches, estimate driving behavior, and detect drunk drivers [63]. To calculate the vehicle's side displacement we use the following equations. It can be used to calculate the speed and distance in the forward direction.

$Side\_speed_i = Side\_speed_{i-1} + Accelerometer(side) * dt$

$displacementi = displacementi - 1 + side\_speed_i * dt$

5.2.4 YAW RATE

Yaw rate represents the movement about the z-axis (vertical axis). Yaw rate can be used to calculate the turning angle and lane changes. Gyroscope sensor measures the vehicle's angular velocity around its vertical axis. It is the angular velocity of the rotation, or rate of change of the heading angle. It is commonly measured in degrees per second or radians per second. If the phone Z axis is the vertical axis (g) then the rotation around its Z axis can be used to calculate the vehicle's angular velocity around the vertical axis. To calculate yaw and yaw rate we use the following formulas:

$yaw\_rate = gyroscope(vertical)$

$yaw_i = yaw_{i-1} + yaw\_rate_i * dt$

5.2.5 PITCH RATE

Pitch rate is the movement about the y-axis (side axis). This can be affected by any change in road slope, bumps, and potholes. The pitch rate is calculated using the gyroscope readings in the side direction.

After preparing the data in the data preparation module, sensors' data will be aligned with the vehicle body. The gyroscope readings in the side direction will be used to calculate pitch rate after removing the gyro-drift using complementary filter in Chapter 4. Given that, gyroscope reading will be integrated to indicate the change in the pitch angle.calculation before and after the filtering process. To calculate pitch and pitch rate we use the following formulas:

$pitch\_rate = gyroscope(side)$

$pitch_i = pitch_{i-1} + pitch\_rate_i * dt$

5.2.6 ROLL RATE

The roll rate is the rate of change in the movement around the x-axis or the forward axis. Similar to yaw-rate and pitch rate, roll rate will be calculated using the filtered and aligned gyroscope readings. Roll rate happens when a vehicle is driving on unpaved or uneven road. To calculate roll and roll rate we use the following formulas:

$roll\_rate = gyroscope(forward)$

$roll_i = roll_{i-1} + roll\_rate_i * dt$

5.3 ADVANCE FEATURES EXTRACTION

Basic features are the features defined by vehicle dynamics papers that needed to understand vehicle's motion. The basic features can be used to detect and understand more advanced motion dynamics like detecting stops, speed estimation, turn detection, detecting lane switches and detecting bumps and potholes.

5.3.1 STOP DETECTION

Stops can be detected using GPS and OBD by considering vehicles to be in stopping mode when the detected speed equals zero. GPS is not very accurate in low speeds, and driving in a very low speed can be detected as stops using the GPS-speed based methods. The most responsive input will be the accelerometer readings

because they can detect the exact moment when a vehicle stopped moving and the moment when a vehicle starts moving again. Figure 30 shows the stopping pattern recorded by the accelerometer. The pattern shows the three stages of the stopping pattern, deceleration, waiting, and acceleration.

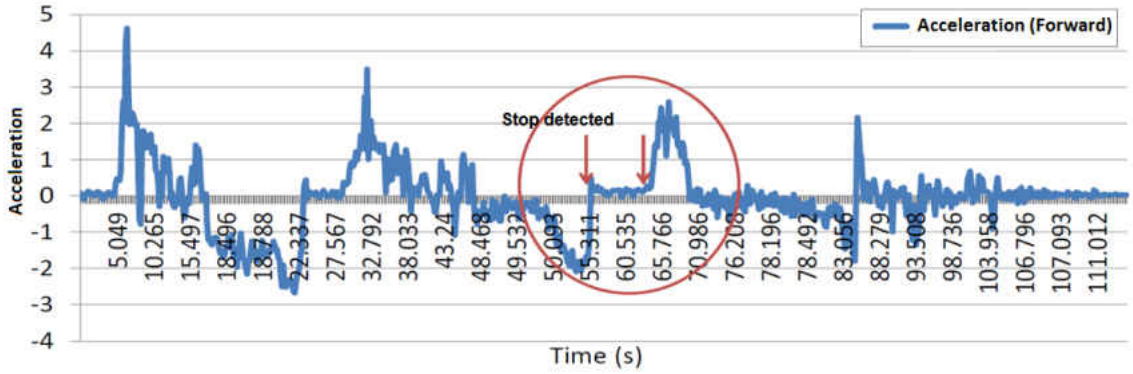
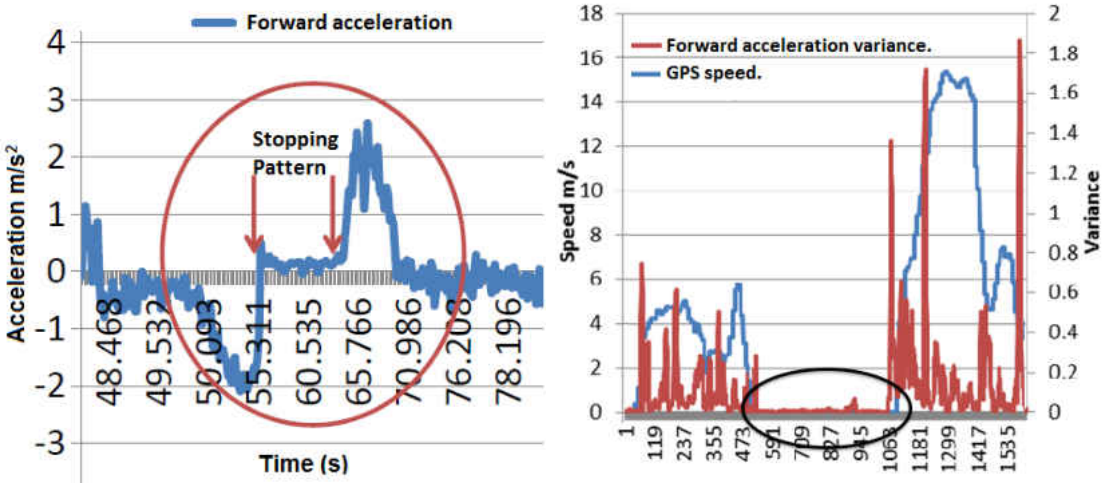


FIG. 30: Stopping pattern

A stop is defined as when a vehicle decelerates, then remains constant with low variance, and accelerates after that which means the car starts moving again as shown in Fig. 31a. This is detected by monitoring the variance of the acceleration feature, when it drops to almost zero, then increases significantly. Figure 31b depicts the effect of stops on the acceleration's variance compared to the ground truth GPS.



(a) Vehicle Stop pattern from raw accelerometer data

(b) Stop detection using the variance of acceleration compared to GPS as ground truth

FIG. 31: Stop Detection

Stops can be detected using variance, patterns or other machine learning techniques like hidden Markov model [18].

5.3.2 SPEED ESTIMATION

A vehicle's speed can be retrieved directly from OBD and GPS. GPS speed is provided by GPS chip which updates the speed every one second. In addition to that the on-board computer OBD provides the real-time speed which is based on the average of the number of wheels' rotations and the size of the wheel. Although speeds from GPS and OBD are very accurate, they do not match each other and they tend to have lower accuracy in low speeds. Figure 32 shows the speed of a vehicle using GPS and OBD compared to the ground truth. Overall, GPS and OBD have high speed estimation accuracy, but the accuracy goes down in the case of very low speeds. In addition to that, GPS speed is not available when the GPS signal is weak, in case of indoors, near tall buildings or through tunnels. Inertial sensors can be used to estimate the speed with higher frequency and detect the small changes in speed that happened in short intervals.

Speed calculation is explained in Chapter 4 as a method of evaluating the coordinates alignment algorithm. The acceleration in the motion direction is used in calculating the speed.

$$speed[i] = speed[i-1] + acc[i] * dt \tag{6}$$

$$distance[i] = distance]i-1] + speed[i] * dt \tag{7}$$

where acc is the acceleration in the forward direction and dt is the time interval since the last reading. Figure 32 shows the calculated speed after applying the data processing and filtering methods in Chapter 4.

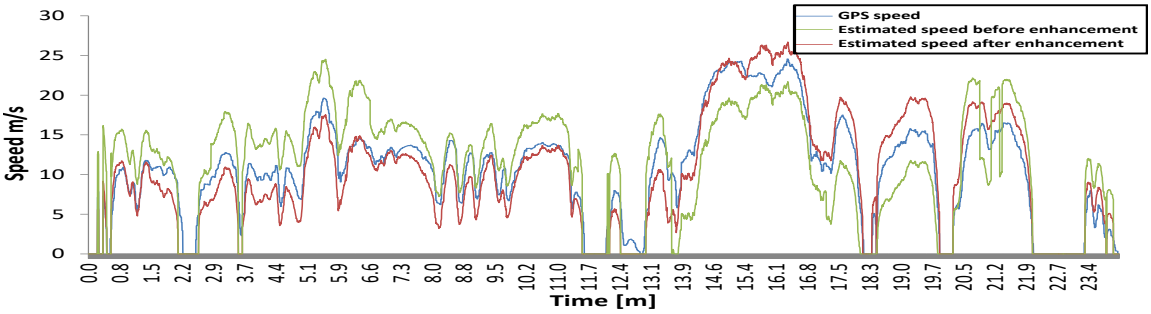


FIG. 32: Speed estimation before and after the enhancement module

5.3.3 TURN DETECTION

Detecting turns is very important feature needed to track vehicle movement. GPS can detect turns by using the turn detection algorithms used by navigation systems which are relative to the road network represented by the geographic database. Inertial sensors can be used to detect turns.

Turns can be detected using different methods and sensors. Figure 33 shows a turn detected using the acceleration in the side coordinate. The system learns the threshold of the acceleration when the car turns and use it to detect the turns.

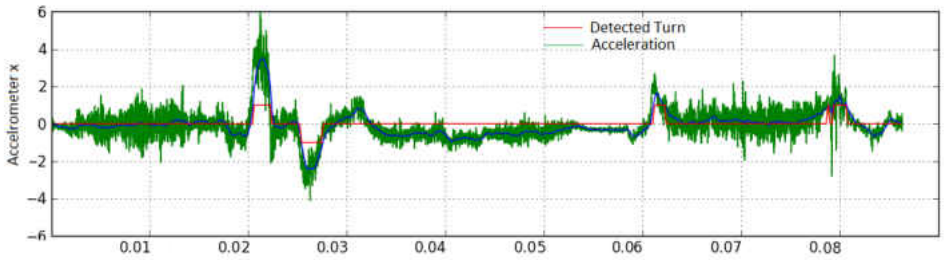


FIG. 33: Turns using side acceleration

Gyroscope is more accurate than accelerometer in detecting turns. Gyroscope are more stable and they only change during turns then go back to zero.

5.3.4 LANE SWITCH DETECTION

Although GPS can detect vehicle's heading and turns, it does not have the enough accuracy to detect a lane switch. Lane switch is very important motion activity that needs detection. Lane switch detection can be important in safety and navigation systems. SenSys uses the gyroscope (vertical) and accelerometer (side) for lane switches detection.

The side acceleration can be used to calculate the side displacement to calculate the number of lanes passed during a lane change. Figure 35 depicts the vertical axis gyroscope readings used to detect 6 of the encountered lane changes.

5.3.5 ROAD-BUMP DETECTION

Detecting road bumps is an important issue in transportation, there are many researches such as [84] that have used smartphones to study road roughness conditions. Such information can be used by governments to monitor the current condition of

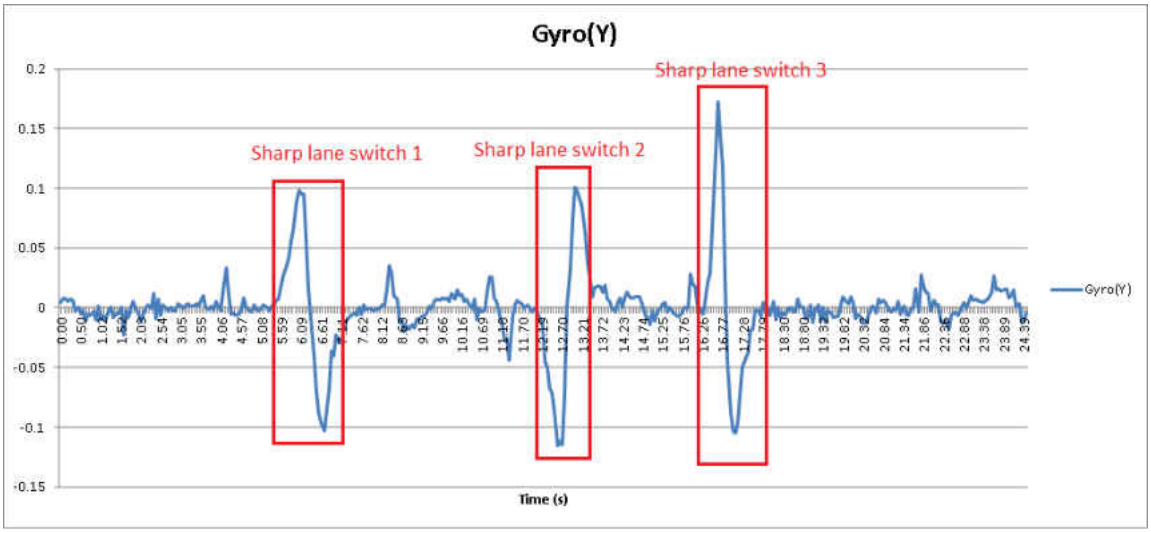


FIG. 34: Lane switches using gyroscope

the infrastructures and help them in making maintenance decisions.

In addition to monitoring the pavement condition, road bumps can be used in different applications for localization, estimating vehicle speed, and detecting vehicle type. Using road bumps, we have built a vehicle type detection application that detect vehicle axles and then measures the distances between the detected axles. Vehicle type detection application is explained in detail in Chapter 10. We also used road bumps and pavement condition as part of a project that detects if two vehicles are driving in the same lane.

Typically, vehicle's horizontal movement is represented by forward and side motions. Vertical acceleration is expected to be stable with a very small variance if the pavement condition is smooth. When a vehicle passes over a road bump, speed bump, or pothole it generates a sudden change in the vertical acceleration reading. This change in the acceleration reading can be used to detect speedbumps and potholes.

SenSys uses Dynamic time warping (DTW) [85] to neutralize the speed effect on the bump detection process. The experiments showed that for SUV and sedan vehicles, when a vehicle passes over a bump, its vertical acceleration exceeds the threshold of 6 m/s^2 for at least 1 second.

Figure 38 shows the accelerometer raw readings for a vehicle driving at a near constant speed on a smooth road while driving over a speed bump. The figure shows how the speed bump generates a shock on the car which affected the acceleration

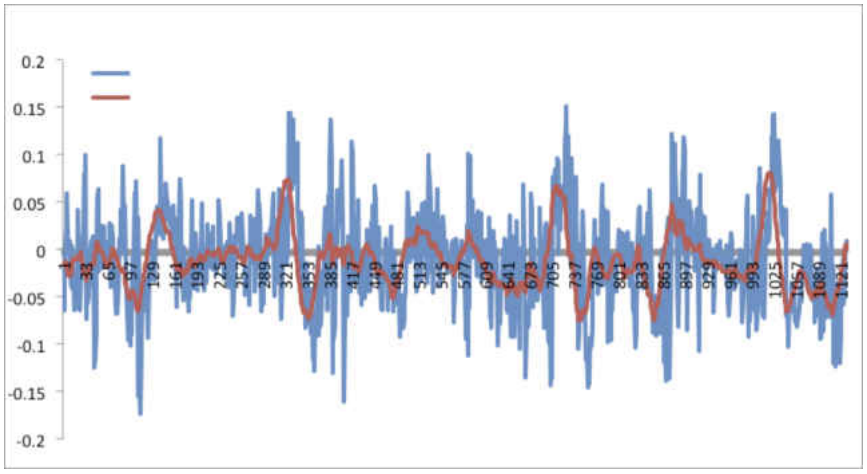


FIG. 35: Detecting 6 lane changes events in one trip.

FIG. 36: Pothole

on all directions specifically the vertical acceleration. In an experiment testing the bump detection using the vertical accelerometer, we used 3 different cars to drive over 6 different speed bumps with different widths and heights. The system was able to detect 100% of the bumps from the data collected.

FIG. 37: Speed bump

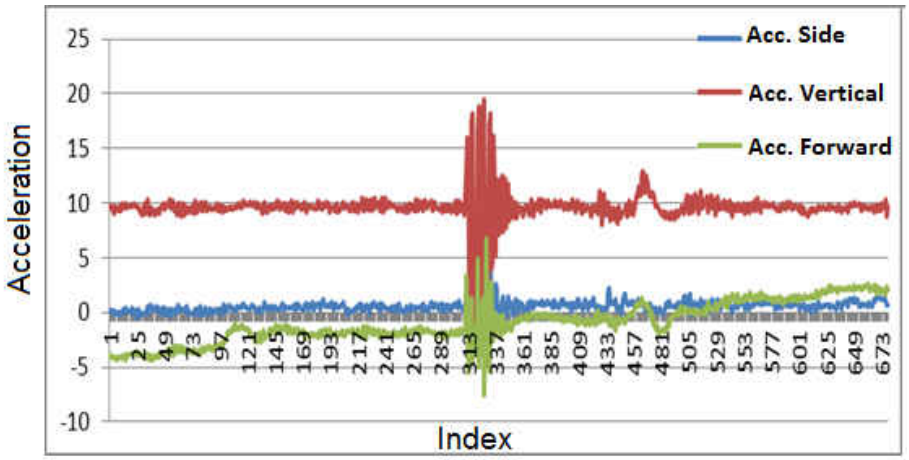


FIG. 38: Raw accelerometer readings using

CHAPTER 6

SENSYS APPLICATION PROGRAMMABLE INTERFACE (API)

In this chapter we explain the application interface, the methods provided by the SynSys library, and some applications built on top of the the SenSys framework. SenSys provides an interface to be used by developers and give them the ability to select the sensors to be used. Developers can select the sensors they want to enable and the data sampling rate. The framework gives the access to the raw and filtered data as well as the ability to add more features to the framework. Developers and researchers can utilize the data provided by the framework to build many types of applications. To test the framework and its features, different applications have been built on top of the SenSys framework and were embedded in the application interface to be used by other developers. The rest of this chapter will focus on the methods provided by the API and some of these applications.

6.1 APPLICATION PROGRAMMING INTERFACE (API)

This section describes the methods and the API calls provided by the SenSys library. SenSys API calls are divided into three groups, the data collection methods, the vehicle dynamics methods, and the add-in application methods. The data collection methods are responsible for communicating with the sensors that are collecting the data and accessing their collected data. This will enable the developers to access each sensor and retrieve its readings directly. Also, it will allow developers to enable, disable, and configure specific sensors based on their requirements.

6.1.1 DATA COLLECTION APIS

SenSys has a set of APIs associated with OBD, GPS, inertial sensors, camera, and microphone. Those are the most common sensors available in smartphones but other sensors can be added with their own set of methods in future updates.

OBD APIs

Communicating with OBD is a pull-based mechanism where developers send commands to the OBD and wait for the response. At first, the developer needs to establish a Bluetooth connection with the OBD. After establishing the connection, the user specifies the signaling protocol which will be used to communicate with the vehicle. There are five signaling protocols permitted with ODB-II interface, and most vehicles implement only one of the protocols. SenSys detects the protocol automatically and starts the communication process with the vehicle's engine control unit (ECU). The signaling protocols are SAE J1850 PWM used by Ford motor company, SAE J1850 VPW used by General Motors, ISO 9141-2 used by Chrysler, European, and Asian vehicles, ISO 14230 KWP2000, and ISO 15765 CAN. As of 2008 all vehicles sold in the US are required to implement CAN as one of their signaling protocols.

The OBD-II standard defines a list of parameter identification numbers (PIDs) that can be sent to the ECU for requesting various data. Once the connection has been established and the protocol has been selected successfully, the OBD is ready to respond to requests from the phone. Table 8 shows a list of API calls that are provided by SenSys to communicate with the vehicle's ECU.

TABLE 8: SenSys OBD APIs

API call (import SenSys.OBD.*)	Description
bool SenSys.OBD.isConnected()	Returns true if the connection is on and false if the connection is off.
int SenSys.OBD.connect(rate)	Establishes a new connection. The "rate" parameter is the number of requests per second.
int SenSys.OBD.stop()	Stops the current connection.
int SenSys.OBD.getProtocol()	Returns the signaling protocol.
double SenSys.OBD.getSpeed()	Returns the current speed in MPH.
double SenSys.OBD.getRPM()	Returns the engine revolutions per minute.
double SenSys.OBD.getFuelLevel()	Returns the current fuel level.
double SenSys.OBD.getPID(PID)	Enables the developer to send any standard PID.

isConnected() returns TRUE if the Bluetooth connection between the phone and the OBD is on, and FALSE if the connection has not been established yet. *connect(rate)* is used to establish the connection with the OBD and set the signaling

protocol, it takes the reading rate as a parameter. *stop()* is used to stop the Bluetooth connection between the phone and the OBD. *getProtocol()* returns the signaling protocol used by the vehicle. *getSpeed()* returns the current vehicle's speed in MPH. getRPM() returns the engine revolutions per minute. *getFuelLevel()* returns the current fuel level of the car, but this method does not work in all cars because it is not one of the OBD-II standard PIDs. Other PIDs can be called using the method *getPID(PID)*.

GPS APIs

Android's location manager provides a set of methods to communicate with the GPS on the phone. SenSys uses this library to communicate with the GPS and then provides the following methods to present the retrieved data. To access the GPS data using SenSys you have to import the GPS library listed in Table 9.

TABLE 9: SenSys GPS APIs

API call (import SenSys.GPS.*)	Description
bool SenSys.GPS.Enabled()	Returns true if the GPS is on.
int SenSys.GPS.start()	Enables the GPS.
int SenSys.GPS.stop()	Disables the GPS.
double SenSys.GPS.getSpeed()	Gets vehicle's speed in m/s.
double SenSys.GPS.getTime()	Gets GPS time in a Unix time-stamp format.
double SenSys.GPS.getLongitude()	Returns the current longitude.
double SenSys.GPS.getAltitude()	Returns the current altitude.
double SenSys.GPS.getLatitude()	Returns the current latitude.

isConnected() can be called to check if the GPS is on or not. It returns TRUE if its connected and FALSE if it is not. If the GPS is not on, you can start it using the *start()* method which enables the GPS on the phone and starts listening to it. *stop()* is used to stop the GPS and disable it. *getSpeed()* gets the vehicle's speed in m/s and *getTime()* gets the GPS time in a Unix timestamp format. The methods *getLongitude(), getAltitude(), getLatitude()* retrieve the current longitude, altitude, and latitude of the moving vehicle. Android's GPS updates its readings with 1Hz frequency.

Inertial sensors APIs

Inertial or motion sensors are important part of SenSys framework, because using them will avoid using the OBD that is not available with most drivers and the GPS which has some accuracy and availability issues as discussed earlier. The three motion sensors are the acclerometer, gyroscope, and magnetometer. Table 10 shows SenSys API calls for inertial sensors.

TABLE 10: SenSys inertial sensors APIs

API call (import SenSys.motionSensor.*)	Description
setRate(rate)	Sets the reading rate.
read(senseroName) Sensors name: accelerometer.[axis] gyroscope.[axis] magnetometer.[axis] X,Y,Z,Forward,Side,Vertical	Reads the sensor in a frequency specified by the *setRate()* method.

Developers can use SenSys to either access the sensors' raw data or access the vehicle's aligned readings. *setRate(rate)* specifies the rate of the readings (number of readings per second). The method *read(sensorName)* takes the sensor name as a parameter which consists of the sensor type and the coordinate like accelerometer.X for the phone's X axis or accelerometer.Forward for the vehicle's forward axis. The sensor name could be any of the following combinations:
accelerometer.X,Y,Z,Forward,Side,Vertical
gyroscope.X,Y,Z,Forward,Side,Vertical
magnetometer .X,Y,Z,Forward,Side,Vertical

Other sensors

The camera and microphone can be used in many types of applications, and accessing them is easy using SenSys framework where developers can start recording the video and the audio with a simple command. Tables 11 and 12 list the camera and microphone API calls that enable the developers to easily access the microphone and camera.

TABLE 11: SenSys camera APIs

API call (import SenSys.camera.*)	Description
camera.video.startRecording(fileName)	Starts recording the video into a file
camera.video.stopRecording()	Stops the recording and save the file.
camera.takePic(filename)	Takes a picture and save it in the file
camera.setCamera(camera)	Sets the camera he parameter could be front or back

TABLE 12: SenSys microphone APIs

API call (import SenSys.microphone.*)	Description
microphone.startRecording(filename)	Starts recording the audio into a file
microphone.stopRecording();	Stops the recording and save the file.

Future updates

SenSys can accommodate new sensors, where developers can create the needed classes to communicate, read, and filter new sensors data and add those methods to the SenSys library.

6.1.2 VEHICLE DYNAMICS APIS

The vehicle dynamics module provides different APIs that sense the vehicle's motion dynamics. Table 13 shows the APIs that retrieve the basic features from the motion sensors:

TABLE 13: SenSys basic dynamics APIs

API call (import SenSys.microphone.*)	Description
SenSys.getYawRate()	Returns current yaw rate.
getPitchRate()	Returns current pitch rate.
getRollRate()	Returns current roll rate.
getVerticalRate()	Returns the current vertical acceleration.
getLateralRate()	Returns the current lateral acceleration.
getForwardRate()	Returns the current forward acceleration.

SenSys also provides more advanced features that understand the motion of the vehicles in the road. Table 14 shows the advanced features APIs.

TABLE 14: SenSys advanced features APIs

API call	Description
SenSys.getCurrentSpeed(Sensor)	Returns the current speed in MPH.
getTurningAngle()	Returns the vehicle's turning angle
isMoving()	Returns true if the vehicle is moving.
getSwitchingLane(seconds)	Detects if the vehicle is switching lane and returns the side (left, right)
getPavementStatus(seconds)	Detects bumps and potholes

getCurrentSpeed() returns the current speed in MPH using the available sensors and the inertial sensor. The parameter determines the sensor used to measure the current speed. *getTurningAngle()* returns the vehicle's turning angle in degrees. *isMoving()* returns TRUE if the vehicle is moving and FALSE if a stopping event was detected. *getSwitchingLane(seconds)* detects when a vehicle switches its lane during the last number of seconds and returns LEFT when it does a left lane switch, RIGHT when it does a right lane switch, and FALSE when SenSys did not detect a lane switch. This function will check the interval given as a parameter and look for a lane switching events within that interval. *getPavementStatus(seconds)* looks for a pothole or bump during the last number of seconds given as a parameter. It returns bump, pothole, or none in case it does not find any within the given interval.

The methods explained earlier in this section were enough to detect all vehicle's dynamics. Many applications can be built on top of them. The following chapters will explain some applications that have been built on top of the SenSys and their methods that have been added to the SenSys framework to enable developers to use them in their applications.

6.1.3 APPLICATION ADD-ON APIS

SenSys was built to be expandable, it allows developers to add new methods to the library. We have built four different applications on top of SenSys and then we added their methods to be part of the SenSys library. Chapters 7, 8, 9, and 10 will explain the methods provided by each application.

CHAPTER 7

PARKING SPACE IDENTIFICATION SYSTEM(PARKZOOM)

Precise localization using smartphones in outdoor and indoor spaces is a challenging task. The widely used GPS is not designed for high accuracy applications and yields accuracy levels not sufficient for lane or spot level localization. In addition, errors from inertial sensors accumulate with time due to integration drift. ParkZoom introduces a smartphone based - infrastructure aided parking localization system for estimating (zooming into) the precise parking spot location of a vehicle during traversal in both indoor and outdoor parking lots. On the vehicle side, the proposed method utilizes conventional smartphones for generating and transferring continuous sensor data, such as accelerometer, gyroscope, and compass readings. On the infrastructure side, ParkZoom employs statistical learning of accelerometer signatures, pattern classification of data, constraint propagation, and error correction for accurate parking spot identification.

7.1 INTRODUCTION

Traditional parking solutions are very cumbersome for the user. The driver has to get out of the car, find the parking meter, forecast/input a tentative parking time, put money into the meter (generally coins), get the ticket, go back to the car and put the ticket in the dashboard. More advanced solutions offer the driver the possibility to manually input the spot number where the car is parked, usually painted on the road or on a road sign. These solutions can be further extended and allow users to speed up the entire parking process with phone payment, e.g. the service offered by companies such as ParkMobile or ParkNow. However, the user still has to know the parking slot number in which the car is parked as the parking operator or system is unaware of the location of the car until the user inputs that information in the system, either with a call, SMS, or web form. A straight-forward solution to vehicle's parking spot identification would be the use of phone's GPS readings, widely exploited in other localization services, e.g. Origin-Destination Navigation, or geo-tagging of pictures.

The problem of parking spot localization is unique in that parking spots are small in size and each parking spot is located next to several other parking spots, making the spot localization and identification process difficult. Infrastructure-intensive parking solutions offer spot by spot parking detection, thanks to the deployment of spot-level sensors, e.g. Streetline parking sensor networks. Such systems are difficult to scale and require a large investment in infrastructure deployment and support.

We envision a parking system able to detect the precise parking spot where a vehicle is parked, with minimum infrastructure deployment costs and minimum user interaction. Such a system would exploit the unique characteristics and maneuvers associated with parking, leveraging readings from standard smartphone sensors. Sensors such as accelerometers, gyroscope, and orientation are able to explore the physical characteristics and movement in the environment, which can be used to uniquely characterize a spatial point. Our solution, ParkZoom, is based on the ability of smartphones inside vehicles to gather data from its sensors and communicate this information to the infrastructure back end system, as shown in Figure 39. The back end system filters and constrains the data based on the physical layout of the parking lot to return with increased accuracy the position of the vehicle, eventually detecting the parking spot or the zone where the car is parked.

ParkZoom uses only the smartphone's inertial sensors to locate a vehicle's parking spot in a parking-lot. Parkzoom will not use GPS in its location estimation, because we designed the system to be used in indoor parking lots and because the error in smartphones' GPS goes up to 10 meters [86]. Calculating the distance traveled by vehicles has been a difficult problem because of the noise generated by inertial sensors. Existing Android apps like greenMeter [87] calculate the distance traveled using the accelerometer only but the error is significantly high and not practical. Step length or wheel rotations have also been used to calculate speed in case of indoor or robots localization [71]. ParkZoom uses only the smartphone's inertial sensors to locate a vehicle's parking spot using the infrastructure of the parking lot.

7.2 SYSTEM OVERVIEW

There are two main challenges that prevent a straight-forward utilization of position and inertial sensors inside current smartphones:

- GPS errors: Current localization systems, such as GPS and Dead Reckoning, are erroneous and have limited position accuracy. For instance, widely used

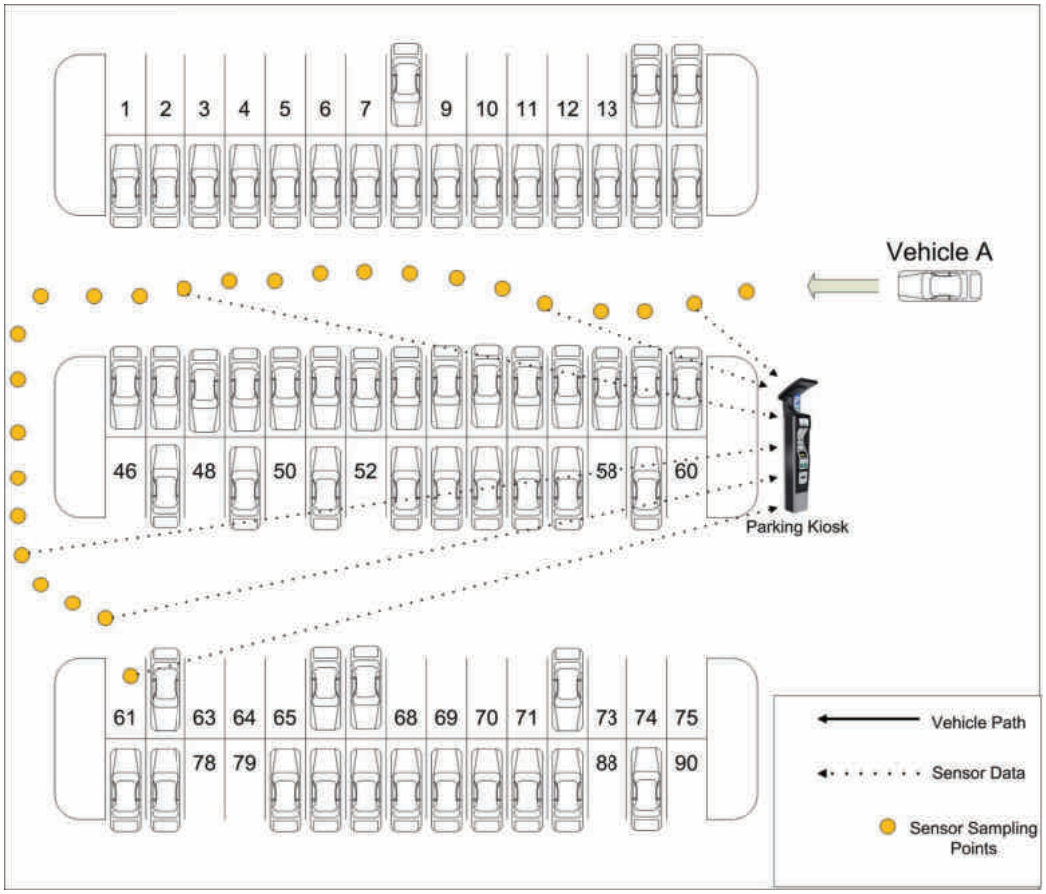


FIG. 39: Smartphone - Infrastructure information sharing

GPS is not designed for high accuracy applications, and yields an accuracy level of 5 meters in open sky setting, 7 meters in young forest conditions and 10 meters under closed canopies [86]. In addition, GPS signal is not available in indoor parking systems.

- Errors from inertial sensors accumulate with time: Inertial navigation systems suffer from integration drift. The position at any point in time is the result of double integrating the acceleration readings. Small errors in the estimation of the latter would be added into increasingly larger errors in the former. In addition, as every new position is based on the preceding calculated position plus the integration of inertial sensor readings, errors increase as the object moves towards its destination. Therefore, position must be periodically corrected [88]. GPS systems or car speedometers are possible candidates, but they do not fit our goal of designing a system that uses only sensors inside today's smartphones.

ParkZoom combines the smartphone's inertial sensor information with constraint propagation techniques and pattern matching algorithms at the infrastructure side. The typical physical structure of a parking lot, represented in Figure 39, is composed of aisles and turning points. Thus, parking maps can be represented as a graph where the edges correspond to turning points. As the information gathered at the infrastructure evolves over time, so does the car along the graph. As a consequence, the localization problem can be divided into two subcomponents: turn detection and distance calculation within an aisle.

The first subcomponent of the ParkZoom system, the turn detection and classification module, has the responsibility to process the signals sent by the vehicle and apply learning-based signal processing techniques for recognizing and classifying the different turns along the vehicle's path. The physical movement through the parking environment can be leveraged to uniquely characterize a spatial point. These unique signatures or "Check Points" can be related to a specific maneuver, e.g. turn [89], or encoded into the pavement via natural irregularities, e.g. bumps or potholes [90, 52, 91], or artificial irregularities, e.g. "Braille-like" pavement stripes [92]. In addition, other readings from alternative phone sensors, such as temperature, pressure, sound or radio signal strength, could also be associated to specific zones in a parking. For simplicity we will consider just "turns" as "check-points".

The second subcomponent of our system, the distance calculation module, is in charge of calculating the distance traveled on the last aisle and specifying the parking spot. However, it does not only calculate the distance at the last aisle but performs the distance calculation at any point in time and stores the value. Aisle length and "Check-point" locations are known by the infrastructure. Before reaching its destination, a car would traverse a random number of aisles and turns until it finds an empty parking spot. "Check-Points" along the route also serve as "learning points" where the real distance traveled can be compared with the calculated value. The difference can then be taken into account in the last aisle distance calculation.

Figure 40 gives an overview of the logic behind our system. Periodically, the smart-phone collects sensor data. The sampling frequency is limited to 50Hz, or one sample every 20 milliseconds, due to software limitations. Sensor data is then processed and segmented. The Turn Calculation module is in charge of detecting the turns, classifying them and returning the turn angle, which will be used to resolve the new aisle. The Distance Estimation module exploits the same sensor data but computes the distance traveled instead. It also keeps track of the errors in the past aisles. The Graph Matching module updates the position of car in the graph according to the information from the Turn Calculation and the Distance Calculation modules. Finally, once the system detects that the car has initiated the parking maneuvers or it has already stopped, the Parking Spot Identification module will estimate the parking location based on the current position in the graph and external constraints.

7.3 TURN DETECTION

In this section, we describe the turn detection algorithm. Parkzoom relies on two stages: (1) identification of the corridor the vehicle is in, through turn detection in a constrained parking lot geometry; (2) calculation of the distance travelled to estimate the exact spot the vehicle is at. Turn detection is an essential component of our system, as it enables us to navigate through the logical parking lot diagram in order to identify the exact corridor in which the vehicle is cruising. It does not assume any infrastructure support and depends entirely on the built-in smartphone sensors. Our proposed solution utilizes a mixture of these sensors in order to identify (1) if there is a turn at a given time; (2) the degree of the turn; (3) the direction of the turn (i.e. right or left). In the following sections, we will discuss these items in

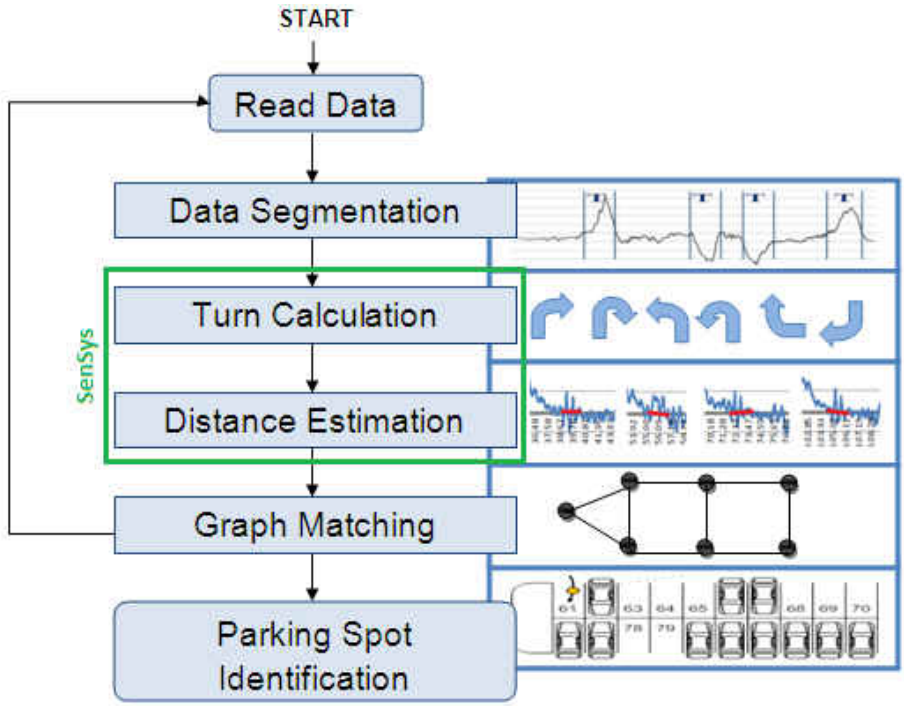


FIG. 40: ParkZoom system overview.

more detail.

7.4 FINDING THE PARKING SPOT

After dividing the parking lot into aisles and turns, we calculate the distance traveled in straight-line (e.g. from the beginning of an aisle to the parking space or the end of the current aisle). We developed an Android application that logs the inertial sensors plus the GPS and the video while driving in the parking lot. Linear acceleration reading is calculated by taking out the gravity effect from the acceleration as given by the Android APIs as follows:

$$V[i] = V[i-1] + a[i] * dt \tag{8}$$

$$D[i] = D[i-1] + V[i-1] * dt + \ * a[i] * dt * dt \tag{9}$$

where A[i] is the linear acceleration at time i, D[i] is Distance traveled at time i, and V[i] is velocity at time i. This will result in impractical error as the error accumulates over time when calculating the speed using these equations only.

GPS, car speedometer, or special sensor installed on the wheels can be used to correct the inertial sensors' readings whenever available. In our case GPS will not be used, because we expect the system to work indoors. Also we cannot use the car speedometer or any external sensors that calculate speed because the system should work on smartphones. Our method uses the fact that the exact location of the turns in the parking lot and the length of every aisle are known. This allows us to correct the accumulated error of the accelerometer. Because we know the geometry of the parking lot, and we can detect the turns accurately, the challenge now is how to calculate the distance travelled from the last turn to the next turn, and when the driver parks.

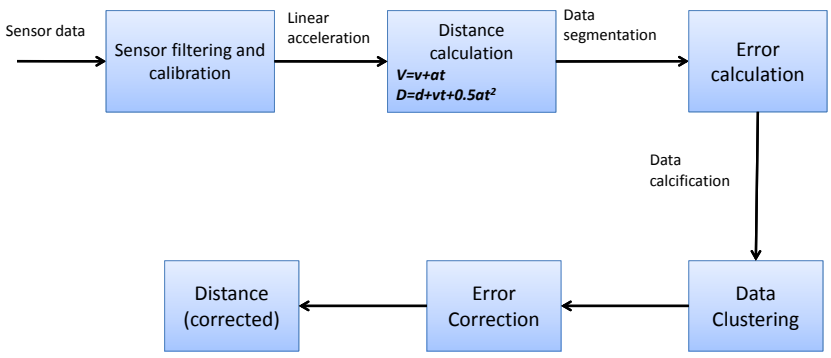


FIG. 41: Distance calculation components.

7.4.1 EXPERIMENT

To minimize distance calculation error, we explored the relationship between the acceleration pattern and the calculated error. To do that we performed our experiment in a parking lot where we know the geometry, and divided each aisle into multiple six-meter segments, we used visual landmarks every six meters and used the video to detect them while driving. After that we drove ten rounds around the

parking lot to collect the data from all the available sensors and we used the video to know the time taken in every segment. At the end of this experiment, we have sets of segments; each segment has its accelerometer data, calculated distance and the actual distance.

7.4.2 ERROR CORRECTION

For the six meter segments, we applied the Error Back-propagation on the results of each segment using the existing trained data of the same segment. Table 15 shows segments 1, 5, and 9 from a nine segmented aisle. It shows the segment calculation before and after the back propagation in different rounds.

TABLE 15: Error back-propagation

Round	S1	S1BP	S5	S5BP	S9	S9BP
Round1	2.7		4.6		4.3	
Round2	3.2	7.11	5.3	6.9	4.9	6.8
Round3	3.4	6.37	5.5	6.2	3.8	4.6
Round4	3.3	5.82	3	5.7	4.9	7.7
Round5	3.2	5.81	5.1	5.78	4.5	5.5
Round6	2.9	5.4	4.9	5.7	4.1	5.4

As it is stated in Figure 41 we use the collected data to calculate the traveled distance, so we trained our system with the collected data, and classified the segmented acceleration into clusters where each cluster contains similar patterns of the segmented acceleration.

The next step is using these clusters to calculate the estimated error for every segment, and since there is no existing landmark to segment the live data into six-meter segments, we divided the accelerometer data into two-second segments. After that we send every segment to the classifier to see which group of patterns it matches. Within the same cluster we compare the test segment with the trained segments to choose the closest pattern to use its error per meter, which we calculated using the estimated and the actual distance, to predict the segment length.

After analyzing the data of the aisles, we found that there are three driving modes that get repeated in every aisle. The first mode is at the beginning of the aisle, where the driver starts to accelerate, the second mode is in the middle of the aisle where the driver drives in almost constant speed which makes the acceleration close to zero, and

the last mode is at the end of the aisle where the driver decelerates. And we found that the error in calculating the distance relates to the mode of the acceleration, and the ratio between the error in mode1 and the error in mode 2 is almost constant, and the ratio between the error in mode 2 and mode 3 is almost constant. So we used this information to get error in the first mode (e1), the error in the second mode (e2), and the error in the third mode (e3).

$$(D1 + D1.e1) + (D2 + D2.e2) + (D3 + D3.e3) = Actual \tag{10}$$

where D1 is the distance calculated in mode one, D2 is the distance calculated in mode two,D3 is the distance calculated in mode three, and Actual is the actual distance.

$$e1/e2 = c1 \tag{11}$$

$$e2/e3 = c2 \tag{12}$$

where c1 and c2 are constants. In the first run, we get c1 and c2 from the clusters in the training set, and after solving the equations (10, 11, 12) we get e1, e2, e3, then we can use them to calculate the distance in the next aisle. At the end of the next aisle, we solve the same equations to get new error values that can be used for the aisle after that.

7.4.3 RESULTS

Figure 42 shows the results after applying the distance calculation technique on some experiments to calculate the traveled distance. In each experiment the car drives in the parking lot, takes some turns before it stops in the parking space. The system calculates and corrects the distance traveled while driving, and gives the distance travelled between the last two turns, the turn of entering the aisle, and the turn of entering the parking space.

The actual distance is the distance from the beginning of the aisle to the parking space, the estimated distance is the calculated distance from the beginning of the aisle after filtering and calibrating the accelerometer data, the corrected distance is the traveled distance from the beginning of the aisle after applying the distance correction techniques on the estimated distance, and the aisle error is the distance corrected using the error per aisle. In the first ten meters, the estimated distance was less than the actual distance by almost 3.3 meters, which goes up to two parking spaces. After

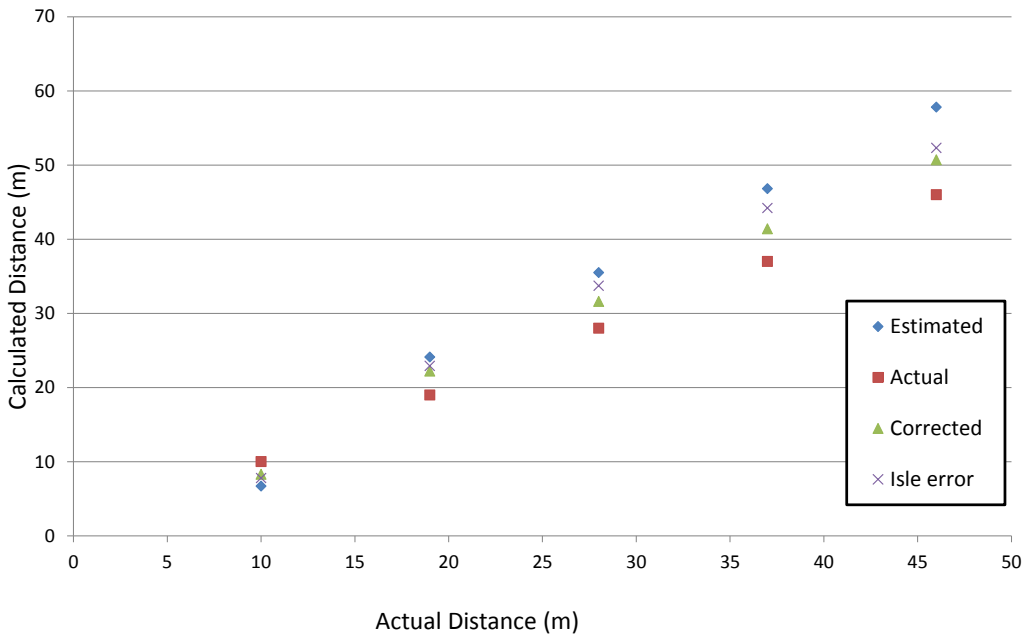


FIG. 42: Calculating the distance from the last turn

correcting the distance, the error became around 1.7 meters. The results in Figure 42 show that the error in the corrected distance slightly increases while driving, while the error of the estimated distance clearly increases by the distance traveled in the aisle. The results also showed that if we want to maintain the error to be less than two meters, then we need to put a landmark every four parking spaces. The landmark can be a small bump on the aisle that can be detected by the inertial sensors, but does not affect the driving behavior of the driver.

7.5 SYSTEM INTEGRATION AND RESULTS

Turn detection and distance calculation are the main components of the system. To test the integration of the system we ran some experiments in a parking lot. We divided the parking lot into turns and aisles, and labeled every parking space in the parking lot.

In every experiment, we enter the parking lot from the main entrance (node A) as

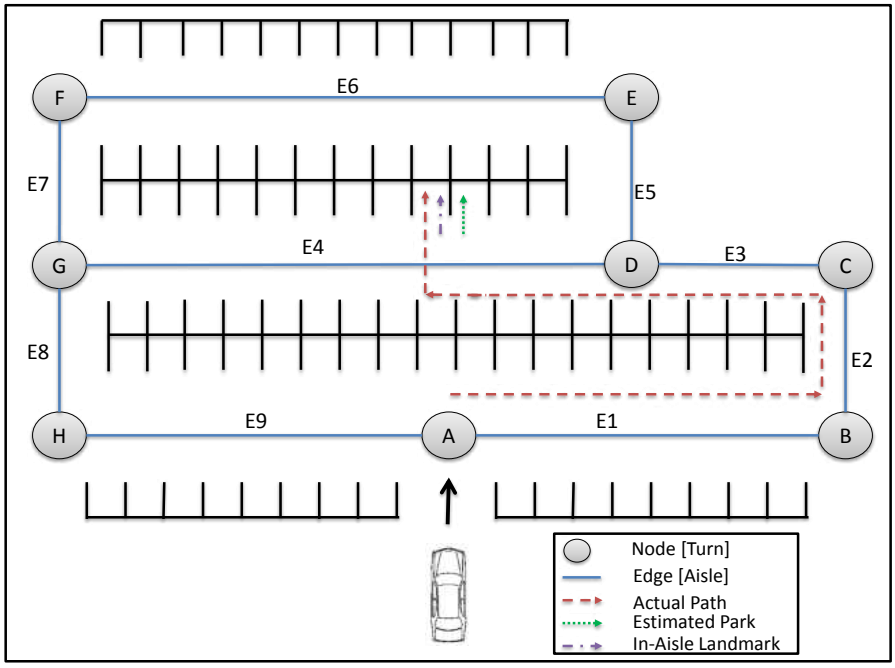


FIG. 43: Parking-lot diagram

seen in Figure 43, and in case of multiple entrances, we assume that, every gate has a unique signature of small bumps that can be detected by the application. Different paths and parking spaces have been taken in different experiments. The system processed the data of every experiment, where the turn detection module specifies the aisle taken, and the distance calculator module calculate the distance traveled on the aisle and specifies the parking space.

7.6 RESULTS

In Figure 44, the car entered the parking lot from the main entrance. From the graph we can see that the car turned right, left, then left again, after that, the car drove for a distance and stopped at parking space to the left. After processing the data, the system was accurate in detecting all turns, and specifying exact aisle of the vehicles' parking spot.

After specifying the aisle correctly, the system calculated the distance driven in the aisle before the car has stopped. Figure 43 shows the estimation after calculating

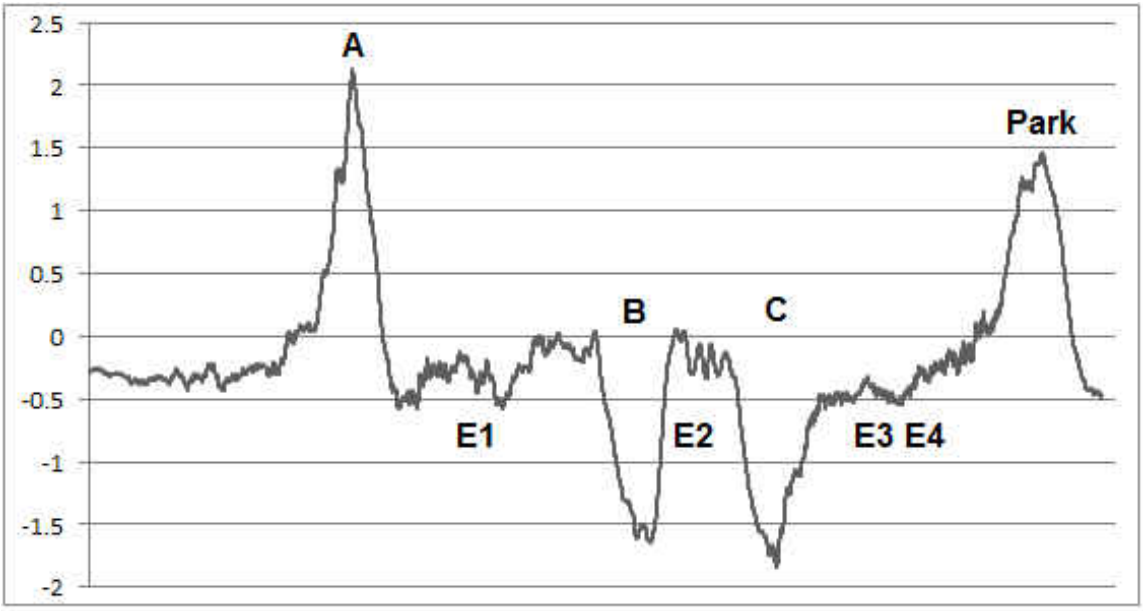


FIG. 44: Accelerometer readings (x-axis)

the distance of the journey. The green arrow shows that the estimated location is one space to the right of the correct location, and the purple arrow shows the estimated location if we place landmarks every 12 meters in the aisle.

7.7 PARKZOOM API

We added ParkZoom to the set of SenSys APIs. The ParkZoom APIs locate a vehicle in the parking lot by detecting the vehicle's turns and calculating distance driven from turn to turn. They also match the detected vehicle movements with the parking lot map to calculate the exact location of the vehicle. The following are the methods used in ParkZoom and added to SenSys library to be used by developers. ParkZoom APIs are listed in Table 16.

getMap() gets the map from the server once it enters the parking lot. *setMap()* sets the map to be used by the application that requires it. *getDistance()* returns the distance in meters driven since the last turn. *getLocation()* returns the location of the vehicle on the map and *setLocation(Location)* sets the vehicle's location on the map.

TABLE 16: SenSys ParkZoom APIs

API call (import SenSys.parkZoom.*)	Description
SenSys.parkZoom.getMap()	Retrieves the map from the server
SenSys.parkZoom.setMap(map)	Sets the map
SenSys.parkZoom.getDistance()	Gets the distance traveled since the last turn
SenSys.parkZoom.getLocation()	Gets the parking spot
SenSys.parkZoom.setLocation(Location)	Sets the parking spot
SenSys.parkZoom.reset()	Resets the location, distance and map.

7.8 SUMMARY

Smartphones applications utilize embedded sensors for activity recognition and positioning. Some activities can be detected with high accuracy, but on the other side, inertial sensors are associated with noise that make them impractical to be used in certain applications. To accurately determine a vehicle's parking space using INS only, we used the parking lot infrastructure to overcome the acceleration noise problem. Turn detection locates the cars on turns, while error-corrected distance calculation and matching with the parking geometry allows ParkZoom to identify the parking spot. Real-world experiments with data collected from two simple and regular parking geometries showed good accuracy in detecting turns and parking-space determination.

CHAPTER 8

IN-LANE COMMUNICATION SYSTEM (INLANECOM)

8.1 INTRODUCTION

The smart cities concept is getting rising attention in the last decade, which has led to more research and development in the field of Information and Communication Technologies. An important aspect of these technologies is the intelligent transportation systems like Vehicle to Vehicle communication (V2V). V2V applications assume general broadcast messages; however, there are several V2V scenarios that would require exchanging information between vehicles on the same lane. For example, in the case of an accident blocking a lane, cars on the front of the blocked lane can notify the cars on the back to change the lane. In addition, cars on the other lanes can be notified to expect traffic coming from the blocked lane. Exchanging such messages will give the drivers extra time to prepare themselves to take the right lane and avoid congestion. Another scenario is when a vehicle drives at a low speed because of a mechanical problem; it can use the in-lane communication system to notify the vehicles behind it. Exchanging messages between vehicles in the same lane will open the door for many ITS applications to increase safety, better traffic control, and improved navigation systems. For example, navigation applications can use this information to enhance their systems redirect drivers from jammed lanes. This system will require minimum interaction from the driver which will minimize driver's distraction. Furthermore, the in-lane communication system will reduce the communication overhead when messages are sent to the desired vehicles only.

Enabling the in-lane communication to apply such scenarios and others will require us to identify vehicles in the same lane. The existing technologies are not able to determine if two vehicles are in the same lane. Currently, drivers use GPS on a daily basis. However, GPS still cannot support lane level accuracy [19]. Also, the use of cameras for lane identification has performance problems [93] where the accuracy drops in the bad weather, and it is limited to determine if the vehicle is within its specified lane or is leaving the lane. We envision that supporting lane-level communication can enable a new spectrum of applications, and navigation enhancements.

The limitations in current technologies have led us to propose an in-lane communication framework (*inLaneCom*). In this section, we introduce *inLaneCom*, a framework to facilitate inter-communication between vehicles of the same lane. *inLaneCom* utilizes the inertial sensors of smartphones on the board of the vehicle to extract two categories of features (i.e. road surface features (physical features), and traffic features). The combination of features extracted will represent the signature of the lane and will be referred as lane signature. To perform the *inLaneCom*, smartphones on board of transmitting vehicles collect inertial sensor readings, analyze them, extract features that uniquely express the current lane, and append them to outgoing messages. Receiving vehicles perform a similar task, however, they compare their generated signatures with incoming ones to either accept or reject incoming messages. The rest of this chapter will provide technical details on inLaneCom performance.

Lane features will be extracted from the raw readings of on-board smartphones' sensors (e.g. Accelerometer, Gyroscope), out of which, *inLaneCom* extracts two types of features out of those readings: physical, and traffic features. Physical features represent the impact of the road surface (e.g. bumps, potholes) on the inertial sensors. Traffic features express the status of the current traffic (e.g. average speed, number of stops). Combining these features, *inLaneCom* creates a signature expressing the vehicle's lane. This signature is used to authenticate single lane communication.

We conducted many experiments using different vehicle types (SUV and full size) driving more than 335 miles both in the city and on highways. The data collected were used to generate lane signatures and to find the similarities between them. Despite the variations of roads, vehicles, and drivers *inLaneCom* was able to detect vehicles within the same lane with high accuracy.

In summary, this work makes the following contributions:

- Identify unique physical and traffic characteristics that uniquely distinguish road lanes.
- Develop a novel solution for lane detection, and segmentation for V2V communications using smartphones.
- Evaluate *inLaneCom* using in-city and highway driving. Results show 94.5% accuracy of lane detection.

8.1.1 USE CASES

Why is in-lane communication important? We envision in-lane as an important extension to V2V technology. Quick mobility in vehicular networks makes it very hard to maintain IPs and send dedicated messages. Therefore, V2V depends mainly on message broadcasting. Furthermore, wasting a portion of sparse communication time to decode an irrelevant message is not a good practice. However, **Is there a need to address a subset of the surrounding vehicles?** This section presents two scenarios where *inLaneCom* can make the difference. Obviously, the smart traffic light scenario does not exist today as smart traffic lights are not available, but ITS researchers have been putting them as a goal that might show up soon. We also envision our work to enhance other scenarios that were limited by the lack of practical lane detection solutions.

The first use case is **emergency vehicles** like police cars or ambulances that use sirens as their warning device through emergency cases. However, limitation shows up in terms of noticeability range and the effect of the noise from outside or inside the vehicle. Studies showed an average of 100 meters noticeability range [94, 95]. In such a distance drivers should take a decision to clear a specific lane for emergency vehicles to pass. In addition, this short distance does not allow drivers to smoothly switch lanes, which might result in traffic congestion. V2V systems starting with a 300m single hop (multihop can do much better) communication as specified by DSRC can deliver a faster message. Furthermore, an in-lane system inform vehicles which lane to clear to avoid incorrect decisions by drivers.

The second use case is **smart traffic lights**. Adaptive traffic control has been a hot research topic for a long time[96]. In-lane communication can provide smart traffic lights with per lane statistics. For example, the left most lane is congested ahead, so increase the stop time for such a lane allowing opposite direction vehicles to pass through.

8.1.2 INLANECOM FRAMEWORK OVERVIEW

To create a framework that enables the communication between vehicles on the same lane, first, we have to find a way to identify vehicles in those lanes. Different lanes on the same road will have a lot of similarities but also will have many differences in terms of road surface and traffic patterns. To find the differences between the lanes,

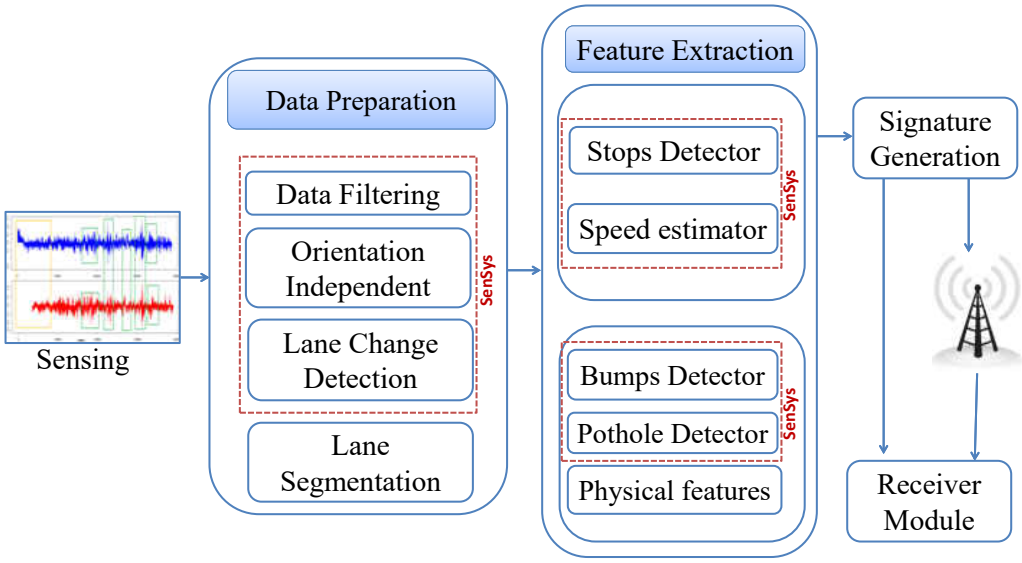


FIG. 45: inLaneCom Framework

we have studied two types of features: physical features and traffic features. Physical features are related to the road surface structure like road bumps and potholes, and traffic features are related to the traffic on the lane like speed, number of stops, time between stops, etc.

All features will be detected and extracted using smartphones, because of the widespread of smartphones and because of their rich set of sensors that can be used to estimate vehicle's motion. Motion sensors like accelerometer and gyroscope are affected by road surface like pothole and bumps and vehicle motion like break and acceleration. Features will be divided into two categories, the features that are affected by road surface and features affected by road traffic.

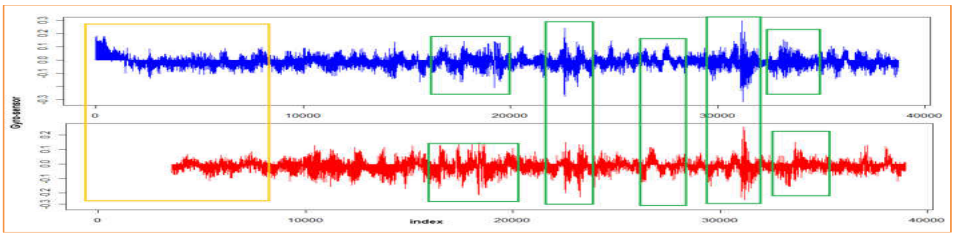


FIG. 46: Gyroscope readings for two different cars driving on the same lane

We ran different experiments to see if sensor readings coming from vehicles driving in the same lane will share similar features. In these experiments, we have a group

of cars driving on the same road. Figure 46 shows the gyroscope readings of different vehicles driving on the same lane. The graph shows that the sensors on the two cars have similar responses to road surface bumps and potholes. In addition, we made some experiments in some busy roads and intersections with different types of cars (e.g. SUVs, full size) and found that the speed and number of stops differ from lane to lane within the same road. Those initial experiments led us to do detailed and through experiments to study the features that can be used to identify vehicle's lane.

In this chapter we propose an in lane communication framework *inLaneCom* that uses the readings of onboard smartphones' sensors to identify vehicles in the same lane and allows messages exchanging between them. *inLaneCom* assumes the availability of at least one smartphone per moving vehicle and targets a reliable communication between vehicles commuting on the same lane. *inLaneCom's* design comes in four modules *data preparation*, *feature extraction*, *signature generation*, and *receiver mode* as shown in Figure 45. The following is a brief explanation of the four modules. The first module is the *Data Preparation* module which consists of four components. The first component is the data filtering component that receives the raw sensing data from smartphones and removes their noise (e.g. accelerometer bias, gyroscope drifts) in order to express meaningful road information. The second component is responsible for aligning the phone coordinates to the vehicle coordinates this will be explained in more details in this chapter. The third and fourth components are the lane-change detection and the lane segmentation modules; they are needed in the signature generation to define lane limits and communication range. The second module is the *Feature Extraction* module, which is responsible for extracting all the available features from the data collected. It starts by retrieving raw sensing data from the inertial sensors (e.g. accelerometer, and gyroscope), then extracting their mathematical features in time (e.g. mean, std. deviation), and frequency (e.g. energy, and amplitude of linear acceleration) domains. Moreover, *inLaneCom* uses sensing data to infer two sets of data, physical features and traffic features. The third module is the *Signature Generation* module, which uses all the collected features to generate a unique signature for the current lane, append it the outgoing packet's payload, and initiate the transmission. And the fourth module is the *receiver side*. This module takes the incoming message, extracts their signature from the payload, and matches it to the locally generated signature. If they match the message is accepted, otherwise discarded without further processing. The following four sections

TABLE 17: Accelerometer Bias

Exps	P1.X	P1.Y	P1.Z	P2.X	P2.Y	P2.Z
1	0.064	0.0403	0.137	0.0891	0.061	0.187
2	0.062	0.0398	0.140	0.0921	0.063	0.188
3	0.067	0.0410	0.139	0.0913	0.0594	0.191

will provide technical details on the *inLaneCom* modules respectively.

8.2 DATA PREPARATION

This section explains how to make the raw sensing data ready for feature extraction by adjusting the orientation, removing sensors' bias, and segmenting the data.

8.2.1 DATA FILTERING

Smartphones come with low-cost inertial sensors which are known to be inaccurate and noisy. To get accurate readings from phones inertial sensors, *inLaneCom* filters and removes gyroscope drift, accelerometer bias, and outliers. Filtering smartphone's sensors and preparing them for vehicular use has been discussed in [17] and in Chapter 4 of this dissertation.

The accelerometer has a fixed bias added to the actual readings. This bias has an accumulative effect when used to calculate the speed. To understand the nature of that bias, we applied different experiments that read phones' sensors in stationary mode. Results in Table 17 shows the bias is almost constant per axis per phone.

Gyroscopes reading are used to calculate the angle taken around vehicle's coordinates. This is used to detect turns, lane switches, road bumps and potholes. Although gyroscope is more stable and less noisy than the accelerometer in detecting the changes in small intervals, in the long run, the gyroscope will drift, and therefore needs a calibration. We built the complementary filter in a similar way to [81], which uses the accelerometer readings to calibrate the gyroscope.

Figure 13 in Chapter 4 shows the calculated angle in degrees before and after the drift correction.

The low pass filter was used to remove the outliers in the horizontal accelerometer readings. It is important to preserve the waveform resulting from bumps and potholes as *inLaneCom* uses them in generating lane signatures. On the other hand, *inLaneCom* needs to remove the random noise that might affect speed or angle calculation. Hence, low pass filter is applied to acceleration in the moving and side directions only.

8.2.2 LANE CHANGE DETECTION

In this module, *inLaneCom* monitors data for lane change event. If not found we append the incoming data to a stream that is forwarded to following modules. Upon detecting a lane change event, the previous data is discarded from all system modules, and a fresh stream is initiated.

In this work, the lane is defined as the path a vehicle takes until a lane reset event

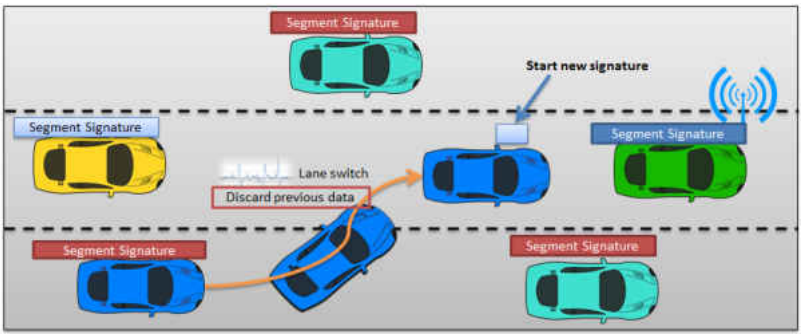
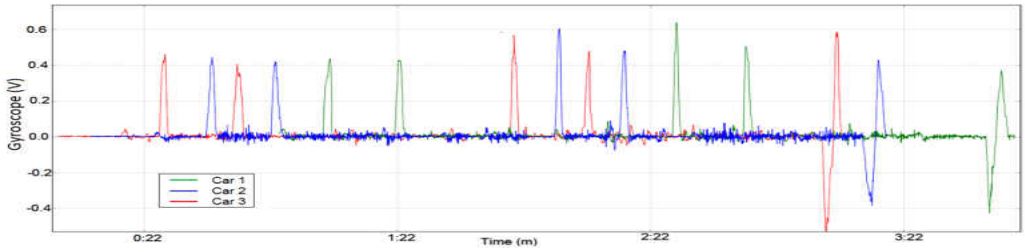


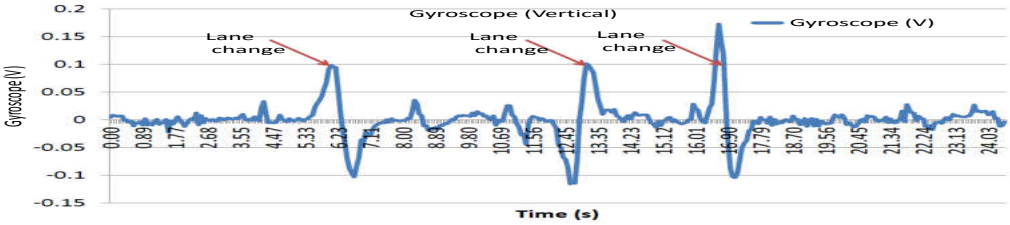
FIG. 47: A vehicle changing its lane. It discards the old signature and starts a new lane signature

is recognized. A lane reset event occurs with one of the following conditions *turns* and *lane switch* as shown in Figure 47. Upon detecting either of them, *inLaneCom* discards the previously collected data belonging to other lanes and starts a fresh feature gathering for the new lane.

Turns and lane switches can be detected using the gyroscope, where its readings are stable until a lane switch or a turn will cause a spike in its readings. *inLaneCom* does not need to differentiate between turns and lane switches because both incidents mean the vehicle will move to a new lane. From a smartphone's perspective, vehicle's turn can be defined as an acceleration in the direction of turning either right or left (x-axis) accompanied by spinning around the gravity direction (y-axis). The system can use the gyroscope to detect the occurrence of turning events and to calculate the



(a) Gyroscope vertical readings used for turn detection. The Figure shows the gyroscope vertical readings for three different vehicles (1 SUV, 2 Full size. From the readings you can tell the cars are following each other.



(b) Lane switch patterns can be detected using Gyroscope vertical readings

FIG. 48: Lane Reset Event

rotation's angle. Figure 48a shows a similar behavior by gyroscope data collected from 3 cars following each other, and taking a turn one after another. The figure depicts how results from 3 different cars show the same behavior representing a turning event.

Smartphone sensors' reaction to switching lanes depends on the driver's habits, which can vary from sharp to smooth. Sharp to moderate lane changes can be detected the same way turns were detected as shown in Figure 48b. The smooth switch can be confusing since the change in the side direction is small. Although our experiments show high accuracy in detecting smooth lane switches, some switches can be very smooth to be misleading. Decreasing the impact of those very smooth ones is explained in our lane segmentation module.

8.2.3 LANE SEGMENTATION

This module receives data stream expressing a single lane. However, if a vehicle drove in the same lane for a long time it will still be represented by only one section. Given that, a single lane can have a different structure in its subsections. For example,

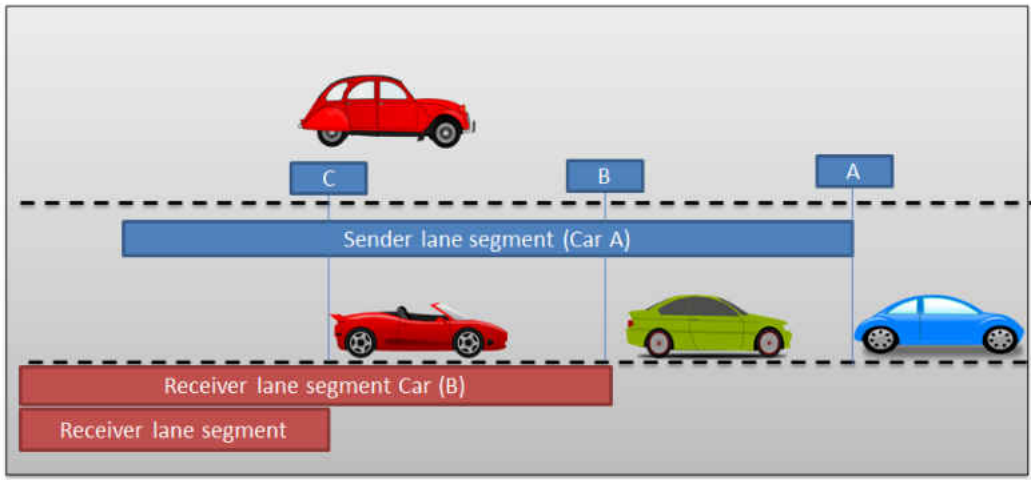


FIG. 49: Vehicles in different locations have different lane segment representations

a 1km lane can have totally different features on its limits based on the quality of pavement, lane elevation, and other road conditions. This leads to poor feature extraction and signature generation for such a long distance. Hence, *inLaneCom* splits the same lane section into fixed shorter length segments (e.g. 200 meters) as shown in Figure 49. Signature generation module will consider the last segment to generate a unique ID for the current lane. This approach provides an up-to-date signature representing the most recent lane features and discard older ones that are likely to provide a poor representation of the current lane. Moreover, having such small comparison segments will quickly recover from an undetected lane reset event, which might occur due to a very smooth lane switching event.

8.3 FEATURE EXTRACTION

To identify a vehicle's lane accurately, we have to determine the right set of features that can represent a lane. To achieve that goal, we experimented using a number of features using smartphones inertial sensors readings and the information that can be extracted from them and found that there are two groups of features that can be used together to identify a lane with a high accuracy. Lane identification using these features is car-type independent and can work with SUVs and small cars.

In this work, *inLaneCom* expresses the lane in terms of two feature categories. *Physical features*: extracted from the characteristics of the lane's pavement and uneven surface. *Traffic features*: extracted from traffic status. These features will

be explained in detail in this section.

8.3.1 PHYSICAL FEATURES

Physical features are extracted from characteristics of lane's pavement structure and uneven surface (e.g. potholes, and bumps). Feature extraction depends on motion sensor responses to the shaking caused by the road surface. Data segments are used to extract the following features: **Mean** of the acceleration in the vertical direction represents the roughness of the road. Potholes and bumps on the surface will shake vehicles in the vertical (gravity) direction, which is detected by the accelerometer. **Standard deviation** and **Variance** of the acceleration, and angular velocity in the vertical direction, they vary per lane regardless of the absolute values. **Energy**, taking the absolute value of the gravity acceleration, the uneven surface affects the acceleration sensed on the gravity direction.

8.3.2 TRAFFIC FEATURES

Drivers have different behaviors in various situations. For example, the acceleration/deceleration patterns vary dramatically between drivers to the extent of being detected by a regular person in the passenger seat. These behaviors are sensed by the smartphone's inertial sensors. Out of which, *inLaneCom* uses the data segments to extract speed-related features; acceleration, and deceleration habits; and the brakes usage features to stop and go.

Speed related features (e.g. Average speed, Maximum, and Minimum speed): Congestion levels have a direct impact on vehicle's speed levels. Our experiment showed that the speeds of vehicles across different lanes vary, especially during rush hours. For example, the left-most lane is usually for speeding vehicles, while right lanes are slow for turns, and exits. Speed features can be very useful in highways with High occupancy lanes (HOV). Our experiments showed that driving on HOV lanes are mostly faster and smoother, and cars in HOV lanes have a higher average speeds and lower number of stops and waiting times.

Acceleration/Deceleration: *inLaneCom* detects vehicle's acceleration, or deceleration through accelerometer readings in the direction of motion. Given the varying acceleration reading from positive to negative, implying forward and backward motion, *inLaneCom* takes a moving average, and checks the acceleration status

while changing between modes. For example, from accelerate to constant or decelerate. Then, *inLaneCom* embeds the number of each mode per time frame in the signature.

Stops related features: In high traffic, vehicles are expected to go through multiple stops. Our results showed that stops in different lanes have different patterns. They might be having a different number of stops, waiting times, and time between stops. Stops can be detected by video analytics [27], or by analyzing the accelerometer data [18]. Using the technique explained in Chapter 5.3.1, *inLaneCom* calculates the waiting time per stop, and time between stops, these two stop related information are meaningful in explaining the nature the current lane's traffic. It is expected that, especially during rush hours, when one car stops, all cars behind it in the same lane will stop. Experiments showed that the number of stops, relative distance, and time between stops are more informative than waiting time, especially at high traffic intersections.

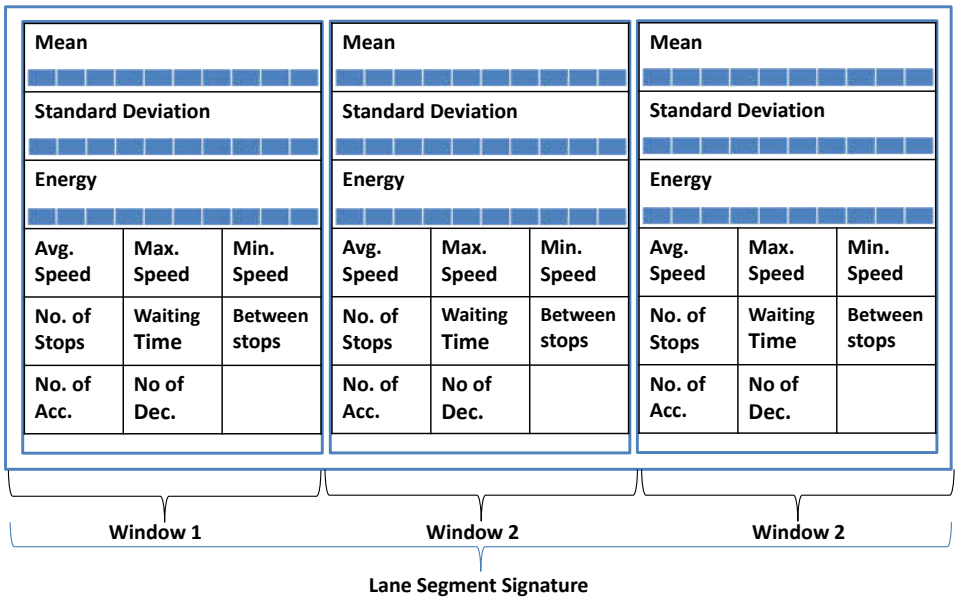


FIG. 50: The lane signature

8.3.3 SIGNATURE GENERATION

Lane signature is generated to represent the last driving segment (e.g. 200m) by the vehicle. Each segment is divided into three equal windows. All features will be extracted from each window separately, then these features will be combined in one

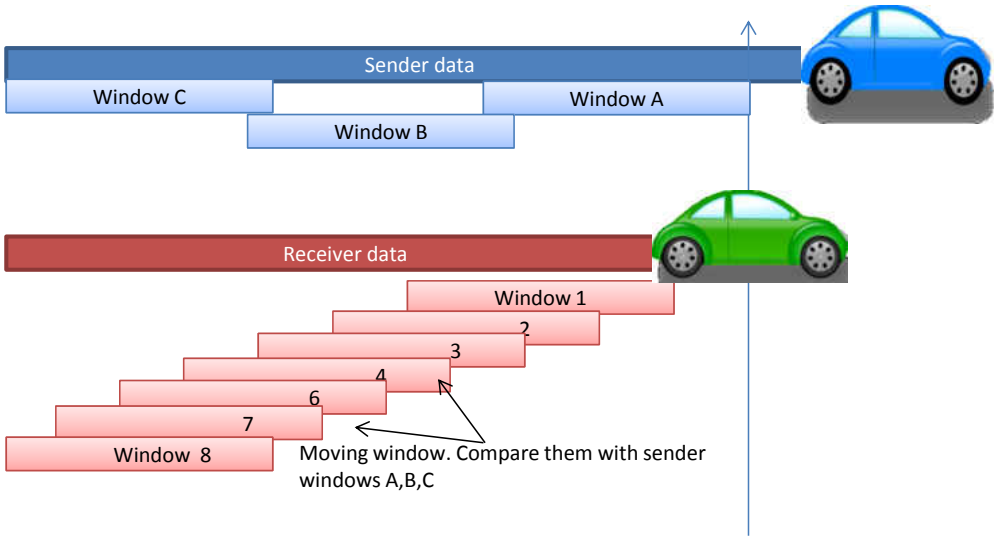


FIG. 51: Dividing the Lane segment into windows

lane signature representing the current lane of the vehicle as shown in Fig. 50. All features explained in the physical features and traffic features subsections will form the lane signature as shown in Figure 50. Mean, standard deviation and energy will be represented as 20 values per window. This will generate a good representation with lower resolution for each window. The number of values has been chosen to keep the signature less than 1K Byte. Since all values are floating numbers, and the size of each number in memory is 4 bytes, then the signature size will be the size of three windows, each window contains the mean which is (20 * 4 bytes) plus standard deviation(80 Bytes), and the energy which is (80 Bytes) plus 32 Bytes for the traffic features per window. The total size will be 816 Bytes for the whole signature.

inLaneCom aims to facilitate smooth communication between multiple vehicles in the same lane. Hence, signatures will be calculated and generated using the same features and sensors on both senders and receiver sides. *inLaneCom* keeps collecting, and calculating signature of the vehicle's lane all the time The signature should represent a portion of the lane that is long enough to calculate and extract features, and short enough to include vehicles in the same. Based on the windowing system used in the application, the last driving segment will be divided into smaller windows, and each window will have its own features which will be processed at the receiving side.

Our experiments show that vehicles have to drive for 60 meters to start calculating representative features. Since the sender will divide the segment into three windows, each segment will be at least 180 meters long. In order to include all vehicles within the communication range, the window was increased to 75 meters, and the segment length will be 200 meters with small overlapping between windows. Upon creating a signature it is appended to the message payload, and transmitted with the outgoing packets.

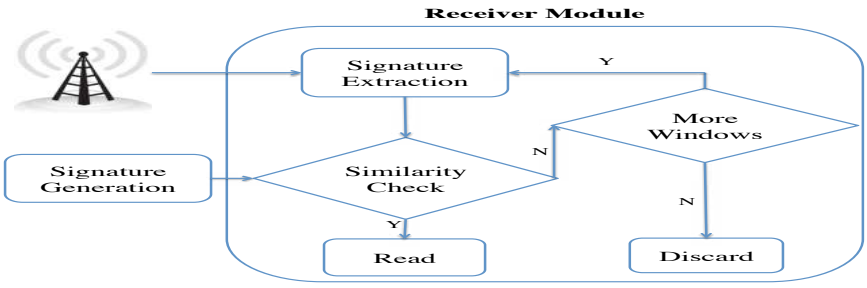


FIG. 52: The Receiver Module

8.4 RECEIVER MODULE

Whenever a vehicle receives a packet, *inLaneCom's* receiver module is initiated to extract the signature, and compare it to available ones. Figure 52 shows details of the receiver module presented in Figure 45.

8.4.1 SIGNATURE EXTRACTION

The received messages contain their lane signature. Hence, this module extracts them for similarity check. Moreover, the signature comparison is done through a moving window with the size of 75 meter that moves 5 meters at a time. This module is also responsible for selecting the right window, then passing them to the similarity check module.

8.4.2 SIMILARITY CHECK

Signatures fed to this module are either received or generated and they contain plenty of features from the accelerometer and the gyroscope data as in Figure 50. *inLaneCom* uses Dynamic Time Warping (DTW) to perform the similarity check [85]. After comparing all incoming signatures, the ones that show minimal normalized

DTW distance are considered from the same lane. Our experiments showed that combining results from both accelerometer and gyroscope yields in better predictions than using any of them separately.

8.4.3 MORE WINDOWS

In the case of signature comparison failure while the received data still have more data to compare, then the window will move 5 more meters and sent back to the signature extraction for further comparisons. If none of them matches, the whole message is discarded.

8.4.4 WRONG DETECTION HANDLING

If a message is misclassified either false negative or positive will soon be corrected by the following ones. On the other hand, the short window range of 200m limits the lane length to handle long roads with different lane structures in various sections. Our experiments show an acceptable performance in taking the correct decision as shown later in the Section 8.5.

8.5 PERFORMANCE EVALUATION

We developed *inLaneCom* as an Android application to collect, and analyze the readings from the phone's sensors. *inLaneCom App* collects the readings from the accelerometer, magnetometer, and GPS data. The App reads data from sensors at 10 Hz frequency. This is enough to get every small change on the phone's sensors. In city roads, the number of samples collected per sensor ranges from 150 samples to 800 samples depending on the speed of the vehicle, the number of stops, and waiting time per stop.
Meanwhile, we collected the vehicle's generated data through OBD-II interface, which is mandatory in all vehicles according to U.S. Law since 1996. ODB-II adapters are plugged into the interface and connected to the smartphones via Bluetooth to provide plenty of vehicle's information, out of which speed was our main concern.

8.5.1 EXPERIMENTS SETUP

We conducted a number of experiments to show the performance of *inLanCom*. In these experiments, we used student volunteers to collect sensing data while driving

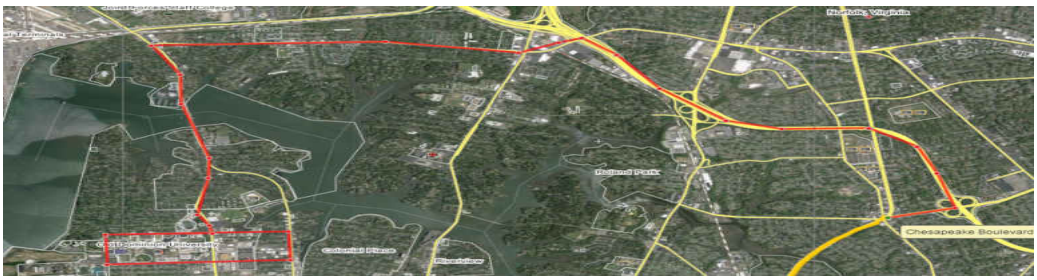


FIG. 53: Experiments held in different types of roads.

vehicles in controlled, and free driving modes. *Free driving* enables volunteers to collect data while their regular commutes for a period of one month. OBD-II adapters were used as ground truth to verify *inLaneCom's* speed, turns, and stop detection mechanisms. *Controlled experiments*, involved different types of vehicles following each other while applying different scenarios that might happen like joining and leaving the lane. The first vehicle sends a message along with its lane signature. Others receive the messages, calculate their own lane signatures, and apply inLaneCom's matching techniques to decide to check whether they share the same lane or not. We conducted different types of experiments that included different types of cars (i.e. SUV, full-size sedan, minivan). The total driving distance in the controlled experiment was 75 miles, part of it is in the red route shown in Figure 53, while the free driving was 299 miles. The experiments were held in different types of city roads and highways. Each vehicle carries a Samsung phone running Android Lollipop. All vehicles have their phones fixed to the windshields in the same position to eliminate the impact of different orientation. To relax the fixed orientation assumption we used SenSys coordinate alignment feature.

8.5.2 FREE DRIVING SCENARIOS

As mentioned previously, we collected sensing information from volunteers' normal commute over a month. Using OBD-II as a ground truth enabled evaluating *inLaneCom* under unconstrained driving conditions. We collected data from 299 miles driving distance over one month, out of which OBD II detected 143 stops, 108 turns, and an average speed of 37.8mph. On the other hand, *inLaneCom* detected 138 stops, 107 turns, and average speeds of 38.3mph with an error of 4.5%, 4.7%, and 5% respectively. Results in the remainder of this section are all derived from

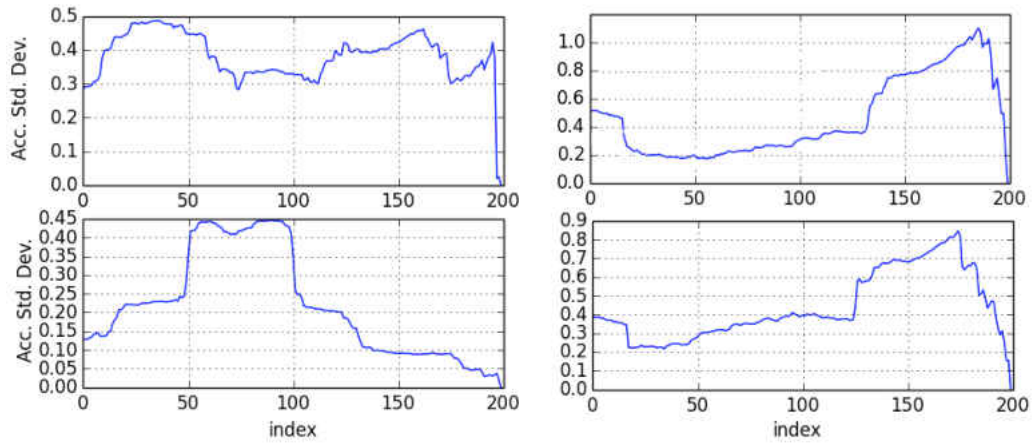


FIG. 54: Accelerometer Reading for the same vehicle/smartphone driving in 2 different lanes of the same road

controlled experiments.

8.5.3 DETECTING LANE CHANGE EVENT

inLaneCom starts it work by lane segmentation as explained earlier through detecting turns and lane switches to define the start, and end of each lane. Detecting turns and switches are done by monitoring the vertical-axis gyroscope readings.

Turning Events: To test the turning detection, we used the data collected from the 299 miles, the system was able to detect 107 turns out of 108 turns with 99.1% accuracy. InLaneCom used SenSys turning detection method to detect turns.

Lane Switching: We drove two types of vehicles (i.e. SUV, full-size sedan), each performed lane switching on different speeds 20 mph, 35 mph, and 50 mph. And the accuracy of detecting the lane switches was 100%, 94%, and 100% respectively. InLaneCom used SenSys lane-switch detection method to detect lane switches.

8.5.4 LANE DETECTION

For lane detection, we have vehicles drove in highways and in city roads.

In City Driving: A vehicle was driven around the city and the university campus on a 2-lane road. The experiment was repeated twice with the same settings, each in

TABLE 18: Performance accuracy in different types of roads

Features	In-City	In-City (traffic)	Highway	Highway (traffic)
Physical	83.1%	84.8%	63.7%	68.7%
Driving	78.2%	86.0%	76.1%	77.5%
Combination	89.8%	94.9%	81.1%	85.4%

a different lane to plot the differences. The collected data was divided to 14 segments, out of which 12 came up to be different (86% accuracy). Figure 54 shows 2 windows per lane, each row represents a lane. The firsts are different, while the seconds are similar across both lanes.

Highway Driving In this experiment, we drove 8 vehicles over the red path shown in Figure 53 for a distance of 36 miles. The collected data were subject to cleaning, and filtration procedures explained earlier. The results show that, on highways especially during rush hours, the traffic features perform better. In city roads, the road physical features are performing well along with the traffic features. In case of high traffic the traffic features perform really well. *inLaneCom*, performs better when the traffic is not normal, for example in case of blocked lane caused by accident, malfunction vehicle, flood, working area, or congested exit. The accuracy gets higher when the speeding and stopping patterns vary from lane to lane, number of stops will increase, and the acceleration pattern changes frequently.

Based on this experiment, *inLaneCom* detected vehicles in the same lane in city roads with an accuracy of 94.9% during rush hours, and 89.8% in regular traffic. The combination of features will be used in our system to give the best accuracy it can get to identify the vehicles in the same lane as shown in Table 18. In-city scenarios provide an environment that is rich in features to grasp. Hence, *inLaneCom* has an acceptable performance in cities.

On highways, *inLaneCom* has an accuracy of 81.1% without traffic, and 85.4% with a high traffic during rush hours as shown in Table 18. As discussed earlier, the accuracy changes from portion to portion based on the physical and traffic situation on the road. In some segments of the highways, because of smooth pavement, and low traffic, there were not enough physical or traffic features to collect. Clearly, traffic

added rich features to the driving patterns, especially number of stops, accelerations, and deceleration patterns. Combining physical and traffic features increased the accuracy of the system. However, *inLaneCom's* accuracy on highways needs to be enhanced to overcome the limited feature environment.

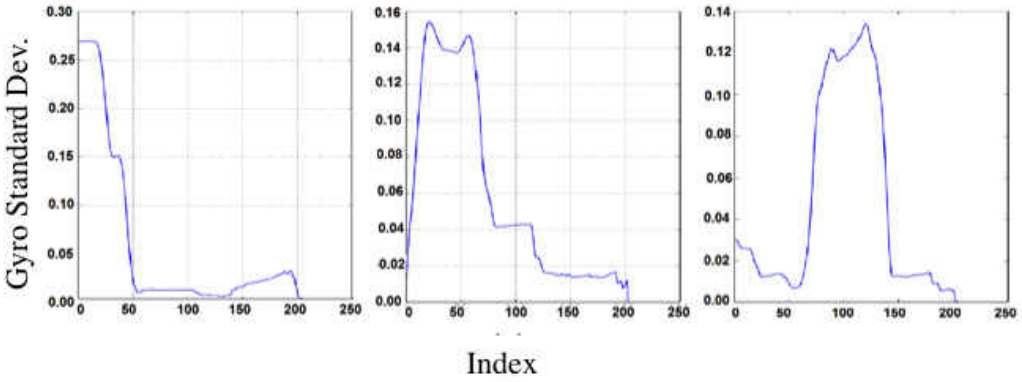


FIG. 55: Standard deviation of the gyroscope data from sender vehicle. Y axis is the gyroscope reading, X axis is sample id.

8.5.5 LANE SIMILARITY

Roads physical features were extracted using mean, standard deviation, and energy of the the inertial sensors reading. Figures 55 and 56 shows the standard deviation of the gyroscope in the side coordinate. Readings represent changes in the spinning angle around the side axis. These values are part of the signature, so the sender vehicle will divide its lane segment into three windows, and send them to the receiver. Figure 55 shows the three windows in the sender lane segment. Figure 56 shows the standard deviation of the gyroscope in some of the moving windows from the vehicle on the back that received the message. Vehicles in Figures 55 and 56 were driving in the same lane. Using dynamic time warping DTW, *inLaneCom* was able to correlate between window 1 in the sender and window 8 in the receiver side, and between the sender's third window and the receiver's seventh window.

8.6 DISCUSSION

Communication: *inLaneCom* utilizes the payload of 802.11 data frame formats [97]. The payload is split into chunks of 2313 bytes to fit in the *network data* field. The signature generation results in 816 bytes signature. In our frames, we

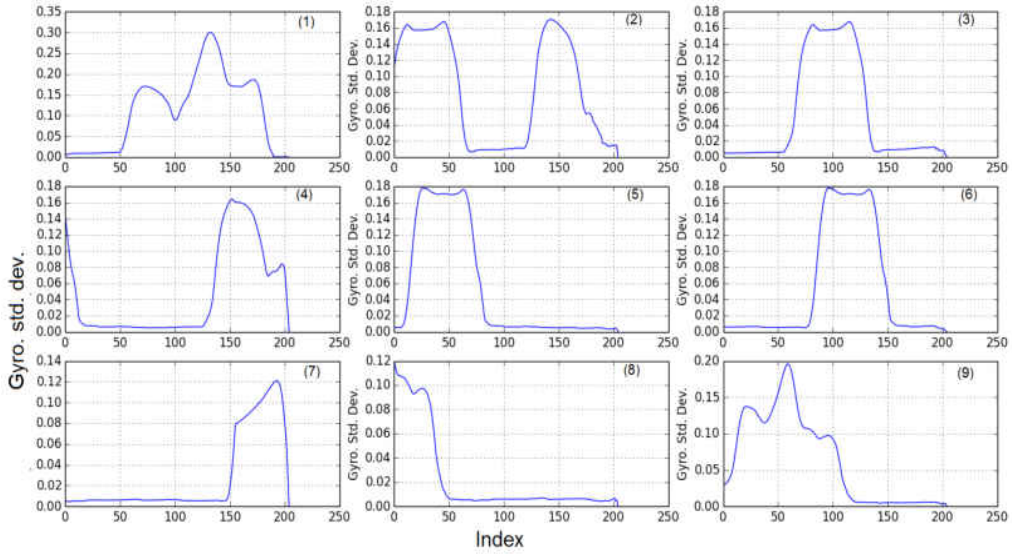


FIG. 56: Standard deviation of the gyroscope data from moving window on the receiver vehicle

append the signature at the beginning of the outgoing message's payload. In this case, we decrease the packet throughput by 36%. However, the remaining amount of the payload is sufficient for the V2V emergency messages.

In the case of non-emergency messages like commercial distribution, messages require more than one packet. In this case, the signature overhead can be significant as in Figure 57, especially if we considered the short communication time period in V2V applications. Hence, we handle these messages in a different manner. In the first packet, we send the signature preceded by 20-Byte SHA-1 hash value of the signature [98]. The remaining packets' payload will only carry the 20-Byte hash value. Figure 57 depicts the effect of this approach on the communication throughput.

Lane Segments size: *inLaneCom* dissects every single lane to multiple segments of smaller size to express the features of the surface. From our experiments, we saw that extracting a single signature for the whole lane can be misleading and provides low accuracy. This shows up clearly if the lane has a different surface structure at the beginning and at the end. We solved this by splitting the lane into segments with different signatures. Our experiments showed a 200-meter segment to give the best accuracy.

Highway Scenarios: Experiments and features used shows *inLaneCom* to favor

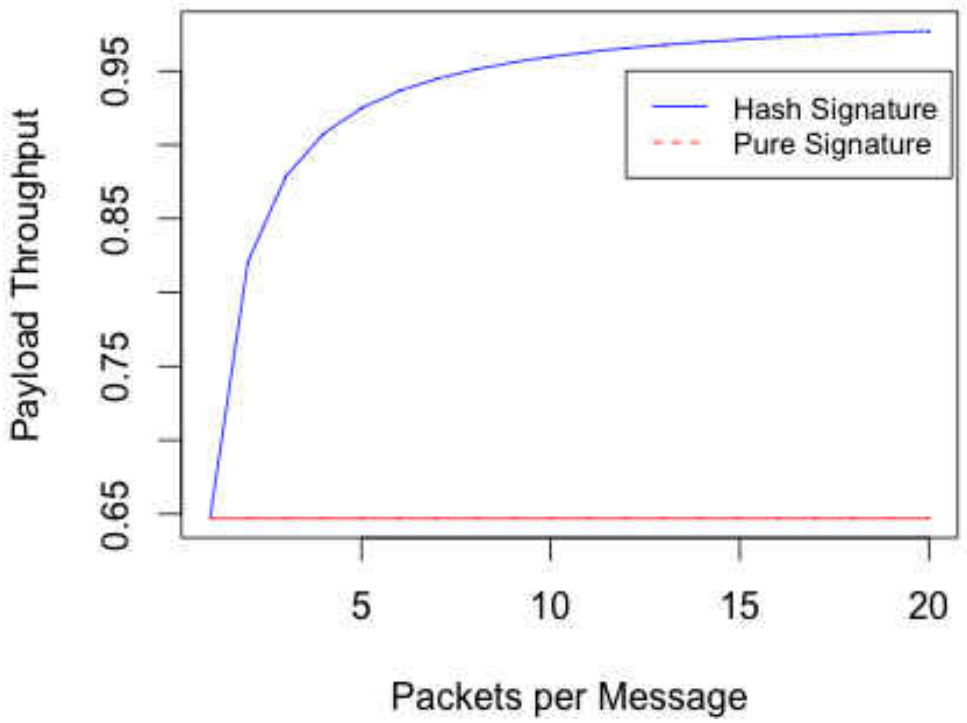


FIG. 57: inLaneCom Communication Throughput

in-city scenarios over highways. The nature of highway's smooth pavement and motion patterns can hardly provide a clear evidence for lane's classification. However, there are candidate features that can increase *inLaneCom's* accuracy on highways. For example, lane curvature can differentiate between different road lanes. It was initially estimated through mathematical models [99], video analysis techniques [100], and recently there are few attempts to get it via inertial sensor readings [101]. Hence, the challenge is how accurate can inertial sensors detect the lane curvature, and whether this will enhance *inLaneCom* on highway scenarios.

8.7 INLANECOM APIS

The in-lane communication application allows vehicles in the same lane to communicate with each other. We used the *inLaneCom* functions to be added to the SenSys framework as APIs. The new set of APIs will expand the usability and functionality of SenSys APIs pool.

Table 19 shows the in-lane communication APIs. *setWindowSize(size)* method

TABLE 19: SenSys in-lane communication APIs

API call (import SenSys.LaneCom.*)	Description
SenSys.LaneCom.setWindowSize(size)	Determines the size of the window in meters
SenSys.LaneCom.signature()	Returns the lane signature for the last window size
SenSys.LaneCom.reset()	Resets calculating the signature.
SenSys.LaneCom.sendTo(Msg,receivers)	Sends the message to either "All" or "Lane"

specifies the size of the window in meters that will be used to calculate the signature. The developer uses *sendTo(message, receiver)* to send the message to either "All" vehicles within the communication range or "lane" vehicles within the same lane.

8.8 SUMMARY

The in-lane communication system proposed a communication scheme between vehicles in the same lane. On-board smartphones are used to detect the current lane and inertial sensors are used to generate a signature for the current lane. The signature is appended to the outgoing messages. Receivers decode the signature and decide whether to accept or deny the message with 95% accuracy. Experimental evaluation depicted how to extract lane's features (physical, and traffic), generate the signature, send it, and compare it at the receiver's side. *inLaneCom* works with different types and sizes of cars. It also works on different types of roads, and showed the best performance in city roads.

Our future work includes expanding the system to establish a communication mean between fixed infrastructure and on-board smartphones, in-lane communication is needed in some places and intersections to enhance navigation, reduce traffic, and increase road safety. In addition to that, we need to explore more features and sensors to enhance highway lane identification accuracy. Since many applications can use the *inLaneCom* framework, part of our future work is to build more systems that use this framework to enhance road safety.

CHAPTER 9

FUEL CONSUMPTION AND CO_2 EMISSION CALCULATOR (GOGREEN)

Calculating vehicle's fuel consumption and emission are important issues in transportation. Table 20 shows that transportation and industrial processes are the major sources of air pollution. Because vehicle emissions are a major source of air pollution, they can impact the quality of air. According to [8] transportation is responsible for about one third of the total emission of CO_2. CO_2 emission can be calculated by knowing the fuel consumption rate and the fuel consumption can be calculated by using the average speed [3, 102, 103].

TABLE 20: Pollution by source [2]

Pollutants	Transportation	Fuel combustion	Industrial processes	Miscellaneous
CO	87.6%	6.0%	5.4%	10.0%
Lead	13.1%	12.7%	74.2%	-
NO_x	53.3%	41.7%	3.7%	1.3%
VOC	43.5%	5.0%	47.2%	4.4%
PM_{10}	25.4%	38.6%	36.0%	-%
SO_2	7.1%	85.1%	7.7%	0.1%

Calculating the fuel consumption can be used to find a relation between the driving style and the amount of fuel consumed. Many researches such as [104, 105] studied the relation between the driving style and fuel consumption. [105] shows that the fuel consumption differences between a calm driver and an aggressive driver can goes up to 40%.

Eco-driving is a driving style developed in the mid 1990s with set of rules and recommendations to drivers. Many research projects are updating the eco-driving rules like the European Ecodriven project (European Campaign On improving DRIVing behavior, ENergy-efficiency, and traffic safety [106]. In summary, the recommendations guide drivers to avoid unnecessary acceleration and breaking and to driving in

low speed when possible. The studies also have shown that adopting eco-driving led to a deduction of 10% to 15% in fuel consumption.

Many projects used the smartphone's GPS and OBD to calculate the fuel consumption and CO_2 emissions. We have built an application based on an accurate model [3] and we used the inertial sensors to estimate the speeds in case if the GPS and OBD are not available to the driver. Also, inertial sensors are known for their ability to detect different driving activities like acceleration, deceleration, and braking. Having the fuel consumption, CO_2 emission, and the driving style visible to drivers will help them to alter their driving behaviors to adopt the eco-driving style which is proven to be cost effective and better for the environment.

9.1 GOGREEN FRAMEWORK

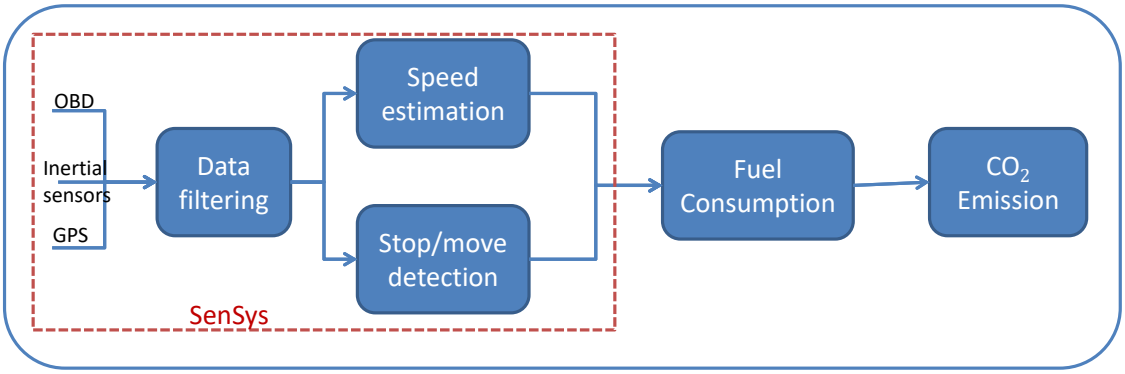


FIG. 58: CO_2 emission calculation components.

GoGreen uses SenSys to read and filter inertial, OBD, and GPS sensors. The processed sensors' readings will be sent to the other SenSys modules to estimate the speed and detect the stops and moves events. The fuel consumption module uses the calculated speed to estimate the fuel consumption which will be used in the CO_2 emission module. We used the model in [3] to calculate fuel consumption and CO_2 emission. The model based its calculations on vehicle speed. We developed an Android app that uses SenSys library to get the speed of the vehicle using inertial sensors, GPS, and OBD and show the average fuel consumption in liter per minute to the driver. Figure 58 shows the framework of the GoGreen app.

9.1.1 FUEL CONSUMPTION AND CO_2 EMISSION CALCULATION

The fuel consumption and CO_2 emission calculations are based on the model in [3]. Their model estimated CO_2 within a 2% error range.

$$FC(t) = \begin{cases} \alpha_0 + \alpha_1 P(t) + \alpha_2 P(t)^2 & \forall \text{ P(t)} \geq 0 \\ \alpha_0 & \forall \text{ P(t)} < 0 \end{cases} \tag{13}$$

$$P(t) = \left(\frac{(R(t) + 1.04ma(t)}{3600\eta_{\mathrm{d}}} \right) v(t) \tag{14}$$

$$R(t) = \frac{\rho}{25.92} C_{\mathrm{D}} C_{\mathrm{h}} A_{\mathrm{f}} v(t) + 9.8066m \frac{C_{\mathrm{r}}}{1000} (C_1 v(t) + C_2) + 9.8066mG(t) \tag{15}$$

where FC(t) is the fuel consumption rate (liter/second) at time t, P(t) is the power exerted by the vehicle driveline (kW) at time t, R(t) is the resistance force. α_0, α_1, and α_2 are vehicle-specific model constants that are calibrated for each vehicle, m is the vehicle mass (kg), a(t) is the vehicle acceleration (m/s_2) at time t, v(t) is the vehicle speed (km/h) at time t, and η_{d} is the driveline efficiency. The constants are different for every vehicle's brand and model. Vehicle constants are explained in Table 21. The example numbers shown in the table are for "2007 Saturn ION".

TABLE 21: Vehicle constant values [3]

Constant	Unit	Example	Description
C_{D}	unit-less	0.42	Vehicle drag coefficient
C_{h}	unit-less	1	Correction factor for altitude
A_{f}	m^2	3.058	Vehicle frontal area
C_{r}	unit-less	1.75	rolling resistance(road surface type)
C_1	unit-less	0.328	rolling resistance(road condition)
C_2	unit-less	4.575	rolling resistance(vehicle tire type

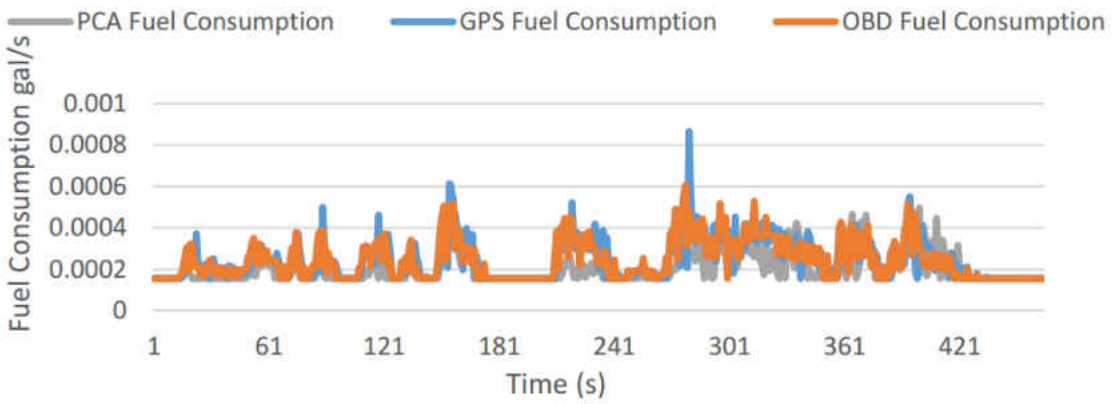


FIG. 59: Fuel Consumption comparison.

Figure 59 shows the fuel consumption calculated using inertial sensors, GPS, and OBD. For this experiment we used the OBD to retrieve the fuel consumption directly and used it as ground truth. Fuel consumption is not standard OBD call but some manufacturers provide it in their new models.

9.2 GOGREEN INTERFACE

The app allows the driver to choose what sensor to use and shows the fuel consumption rate per minute for the last five minutes as shown in Figure 60.

At the end of the trip GoGreen views trip summary that shows the average speed, fuel consumption rate, distance, time, and the total fuel consumed in that trip as shown in Figure 61.

9.3 GOGREEN APIS

GoGreen provides a set of APIs into the SenSys framework which enables ITS apps developers to directly read the current fuel consumption and CO_2 emission. Table 22 shows the list of GoGreen APIs.

getCO2emission(time) gets the amount of CO_2 emission in milligrams. getFuelConsumption(time) returns the total amount of fuel consumed within the last number of seconds sent as a parameter in gallons. reset() resets the fuel consumption and CO_2 emission to zero to get ready for a new trip.

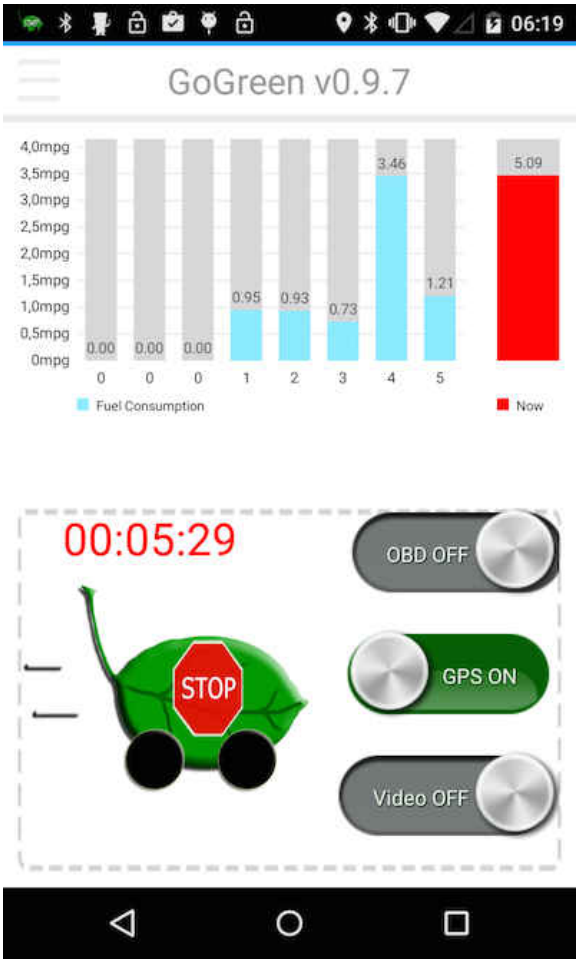


FIG. 60: GoGreen interface.

TABLE 22: SenSys GoGreen APIs

API call (import SenSys.GoGreen.*)	Description
SenSys.GoGreen.getCO2emission(time seconds)	Returns the CO_2 emission in milligrams during the last number of seconds.
SenSys.GoGreen.getFuelConsumption (time seconds)	Returns the fuel consumption rate during the last number of seconds.
SenSys.GoGreen.reset()	Resets the calculation of fuel consumption and CO_2 emission

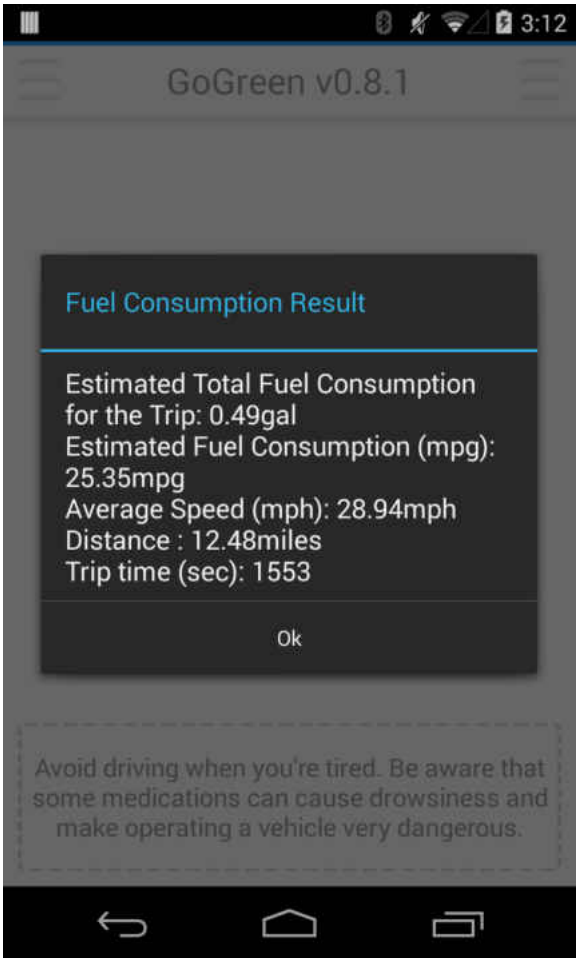


FIG. 61: GoGreen at the end of the trip.

CHAPTER 10

VEHICLE TYPE DETECTION

Although existing technologies provide some important information like vehicle count, queue length, and vehicle weight, they cannot accurately provide some other important information like vehicle's type. Determining vehicle's type is important for enhancing transportation applications, generating traffic statistics and tolls revenue auditing. For example, road tolls need to know number of axles on the passing vehicle to determine the toll fees. The Federal Highway Administration (FHWA) has classified road vehicles into 13 classes. This classification depends on the number of vehicle's axles, tiers and units. Based on the vehicle class, we can tell if the vehicle is a motorcycle, passenger car, van, or truck.

The most common methods used for wheel and axle detection are based on sensors installed in the roadway (pressure detectors) that require construction work to cut the asphalt. But it is known that these sensors suffer from degradation which effects there performance. To overcome the degradation problem, some solutions proposed the use of optical sensors on the lateral side of the road lanes. But the problem with the optical detection technology is its robustness against some weather conditions like heavy rain, mud, and ice. The problems associated with current technologies encouraged us to look for more reliable and cost effective solutions.

In this chapter we propose a new way of detecting vehicle's axles and class using smartphones. The high technology used in smartphones and their rich features in terms of processors, sensors, communication, and memory encouraged the researchers to utilize them in enhancing the existing ITS systems.

This work utilizes smartphones sensors to detect vehicle's axles, wheelbase, and determine its class. This solutions was built on top of SenSys framework which reads, filters, and analyzes smartphone sensors readings.

10.1 VEHICLE TYPE DETERMINATION SYSTEM OVERVIEW

The Federal Highway Administration (FHWA) has classified road vehicles into 13 classes. The classes can be grouped into passenger cars, buses, and trucks. Figure 62 and Table 23 show the FHWA vehicle classes and their description.

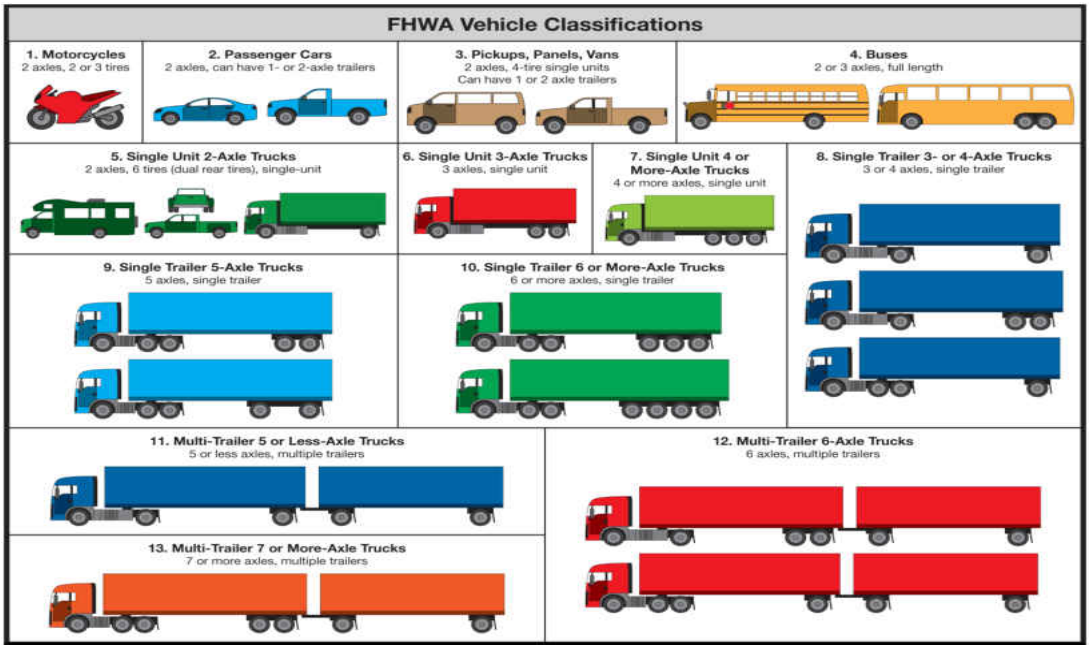


FIG. 62: FHWA Vehicle Classification [7]

To determine vehicle's class we need to know the number of axles and the distances between them. The system uses the shocks generated when a vehicle passes over a speed-bump. SenSys framework which will be explained in the next section is used to detect the bumps which will be used later to calculate the wheelbase. Wheelbase is the distance between front and rear axles of a vehicle as seen in Figure 63. Figure 64 shows how the speed-bump affect the vertical acceleration of a vehicle and how its axles can be detected using the vertical acceleration. When the front axle passes over the speed bump it generates a vertical acceleration at time (t1) and when the rear axle passes the speed bump, it generates a vertical acceleration at time (t2), knowing the speed of the vehicle which can be retrieved from the SenSys framework or GPS we can calculate the wheelbase using the following equation:

$$Wheelbase = (t2 - t1) * Speed \tag{16}$$

10.2 SYSTEM ARCHITECTURE

In this section we will go through the steps of determining vehicle class. The two main steps to do that are axles' detection and wheelbase calculation.

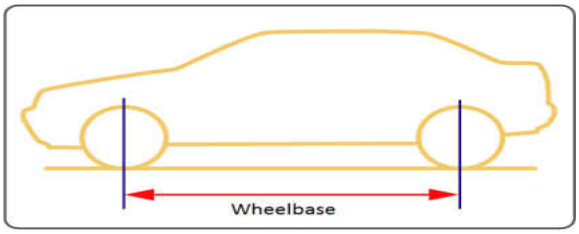


FIG. 63: Wheelbase: The distance between the front and rear axles of a vehicle.

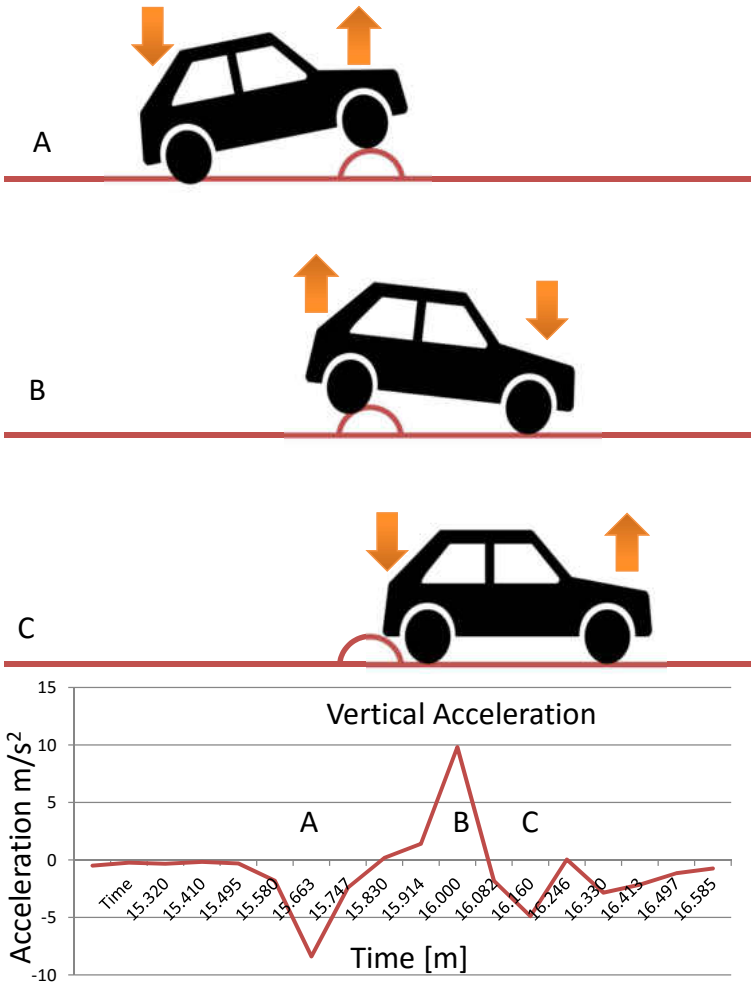


FIG. 64: Speedbump effect on vertical acceleration.

TABLE 23: FHWA Vehicle Classes [4]

Class	Axles	Description
1	2	Motorcycles (Optional). All two- or three-wheeled motorized vehicles.
2	2	Passenger cars
3	2	Other two-axle, four-tire single unit vehicles. Like pickups and vans.
4	2 or more	Buses. All vehicles manufactured as traditional passenger-carrying buses with two axles and six tires or three or more axles.
5	2	Two-axle, six-tire, single-unit trucks.
6	3	Three-axle single-unit trucks.
7	4 or more	Four or more axle single-unit trucks.
8	4 or fewer	Four or fewer axle single-trailer trucks. All vehicles with four or fewer axles consisting of two units.
9	5	Five-axle single-trailer trucks. All five-axle vehicles consisting of two units.
10	6 or more	Six or more axle single-trailer trucks. All vehicles with six or more axles consisting of two units.
11	5 or fewer	five or fewer axle multi-trailer trucks. All vehicles with five or fewer axles consisting of three or more units.
12	6	Six-axle multi-trailer trucks. All six-axle vehicles consisting of three or more units.
13	7 or more	seven or more axle multi-trailer trucks. All vehicles with seven or more axles consisting of three or more units.

10.2.1 AXLE DETECTION

Axle detection is based on bump detection, and bump detection uses the vertical acceleration. Vertical acceleration and bump detection are derived from the SenSys framework. Figures 67, 68 show the vertical acceleration pattern of vehicle passes over a speed bump in different speeds. The figures show that there are some differences between the two patterns caused by the speed of the vehicle. The vehicle in Figure 67 was driving at speed of 6 mph, from the figure, point (A) represent the front axle is over the speed bump, the point (B) when the front axle passed the bump, point (C) when the rear axle is over the speed bump and point (D) when the rear axle passed the bump. All these points are expected to be detected whenever a vehicle with two

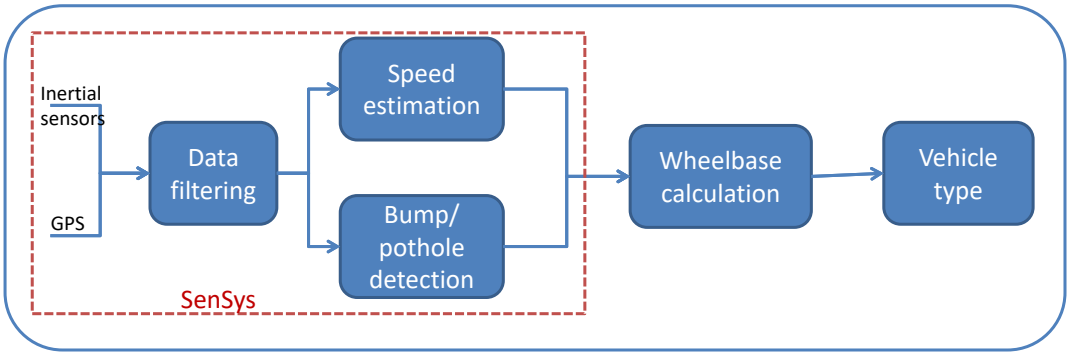


FIG. 65: Vehicle type detection framework.

axles passes over a speed bump. But in case the vehicle is driving with higher speed, and the speed bump is large, then the two points (B) and (C) could be combined where the rear axle arrives the speed bump before the front axle passes it. Figure 68 shows the same car driving over the speed bump with higher speed, and point (B) represents both the front axle leaving the bump and the rear axle passing over it. Both events merged to form a high change in the vertical acceleration. Bumps detection in SenSys framework goes through different modules as shown in Figure 66. First, the system collects the raw acceleration, then filters the outlines and align its coordinates, then it extracts the vertical motion which will be used for bump detection.

FIG. 66: Bump detection modules in SenSys framework

10.2.2 WHEELBASE CALCULATION

Wheelbase is the distance between the front and rear axles. The system records the time when the front axle passes over the bump and then the time when the rear axle passes over the bump. Given the speed of the vehicle, the system can calculate the distance between the two axles. Figures 67, 68 shows the wheel base distance

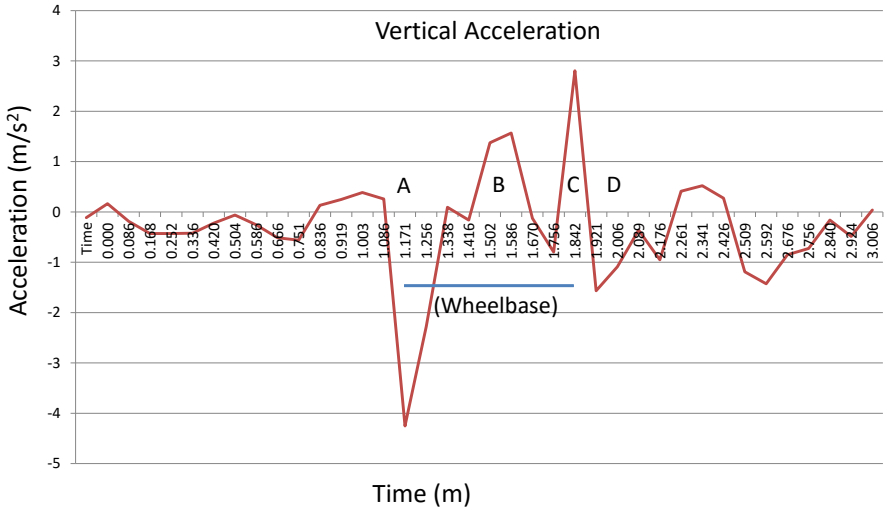


FIG. 67: Wheelbase calculation while driving at speed of 6 mph

between the two bumps. The wheelbase is calculated using the following equation:

$$Wheelbase = (t2 - t1) * Speed \tag{17}$$

Where (t1) is the time when the front axle passes over the bump, and (t2) is the time when the rear axle passes over the bump.

The following section evaluated the two main components of the system (Axle detection and wheelbase calculation) using two different vehicles from two different classes while driving in different speeds.

10.3 SYSTEM EVALUATION

To evaluate the system we ran many experiments, using a sedan vehicle from class 2 and a van vehicle from class 3. We drove both vehicles in different speeds over a speed bump as shown in Figure 69.

Experiments

To test the concept of detecting vehicle types we conducted number of experiments using different types of vehicles. The vehicles used are a sedan Toyota Camry 2000, a Honda Odyssey 2003 Van, 2009 Toyota Highlander and a 2013 Gillig Low Floor bus. Each vehicle has two axles, the sedan has a 2.672 meters wheelbase, the van

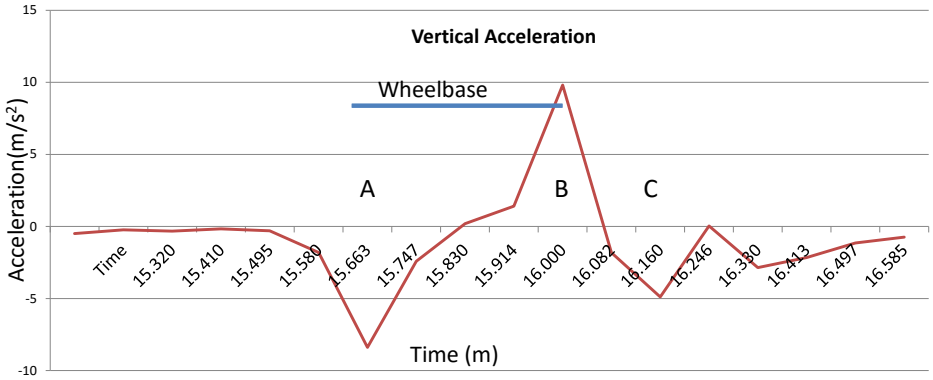


FIG. 68: Wheelbase calculation while driving at speed of 15 mph

has 3.002 meters wheelbase, the SUV has a 2.8 meters wheelbase, and the bus has a 7.07 meters wheelbase. We drove the cars in different speeds over a speed bump as shown in Figure 69. For the bus we rode the bus for more than 85 minutes and analyzed the data for potholes and bumps.

10.3.1 RESULTS

For each run over the speed bump the system calculates the wheelbase and compares it to the ground truth. For the sedan Toyota Camry vehicle with 2.672 meters wheelbase, Figure 70 shows result of seven runs over the speed bump with two different speeds of 5 MPH and 15 MPH. For the 5 MPH wheelbase calculation, the average calculated wheelbase is 2.55 meters with 12 centimeters average error. The average error increased to 17.3 centimeters when driving in 15 MPH speed. The Honda Van has almost similar average error as shown in Figure 71. For the 5 MPH speed, the average wheelbase is 2.87 meters and the average error is 12.5 centimeters. For the 15 MPH speed, the average error increased to 19.5 centimeters with average wheelbase of 3.195 meters. For the SUV, the accuracy of wheelbase calculation was 95.3% with average error of 13 centimeters. The results show that, the average error was almost the same for the two vehicle types.

While riding the bus, the system detected several bumps and potholes. In the

FIG. 69: A Speed bump

experiment, the bus passed over the bumps with different speeds because we did not have a control over the speed of the bus. For every event detected, the system calculated the wheelbase and the class related, the results of this experiment are shown in Table 24.

TABLE 24: The wheelbase calculation in the bus experiment

Event	No. axles detected (2)	Speed (MPH)	Wheelbase (7.07 meter)
Bump 1	2	22.3	6.80
Bump 2	2	15.1	7.26
Bump 3	2	20.6	6.86
Bump 4	2	10.8	7.19
pothole 1	2	14.1	6.88
pothole 2	2	25.4	6.73
pothole 3	2	32.2	6.68

Table 24 shows that bumps and potholes can help detecting vehicle axles, although the experiments showed that speed bumps performs better than potholes. Bus wheelbase calculation gives better accuracy in lower speeds.

Small speed bumps can be placed on toll stations and the drivers could be directed to drive at low speeds. The automatic vehicle type detection can detect the type of the vehicle and calculates its tolls automatically.

10.4 VEHICLE TYPE APIS

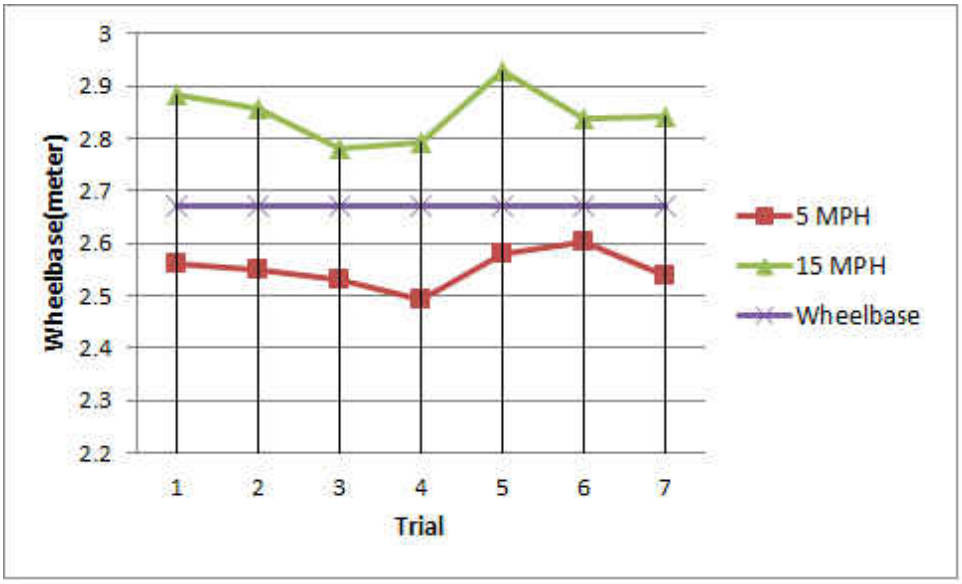


FIG. 70: Wheelbase calculation for the Toyota Camry vehicle with wheelbase of 2.672 meters.

Vehicle type detection is an important issue in ITS and can be used in many applications, therefore, SenSys provides a set of APIs that enable the developers to detect the vehicle's type and other features. Table 25 shows SenSys vehicle type detection methods.

TABLE 25: SenSys vehicle type APIs

API call (import SenSys.vehicleType.*)	Description
SenSys.vehicleType.getWheelBase()	Returns the distance between the front and the back axles.
SenSys.vehicleType.getAxelsCount()	Returns the number of axles detected.
SenSys.vehicleType.getType()	Returns vehicle's type

getWheelBase() returns the distance between the first and the second axles in meters. getAaxelsCount() returns the number of axles detected. getType() returns the class number as declared in [4].

10.5 SUMMARY

Automatic vehicle type detection is an example of traffic applications that can use SenSys. The proposed automatic vehicle type detection uses smartphone inertial

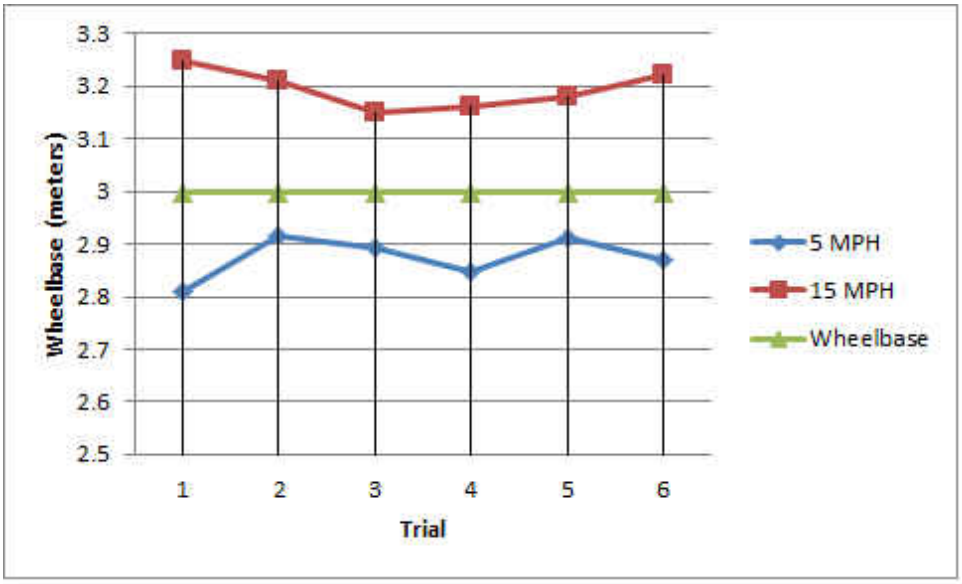


FIG. 71: Wheelbase calculation for the Honda Odyssey vehicle with wheelbase of 3.002 meters.

sensors to detect the number of axles, and then calculates the distances between them to determine vehicle's class and type. The results show that the system was able to detect the number of axles with 100% accuracy and calculate the wheelbase with average error of 12.14 centimeters only for the cars and 24.4 centimeters for buses with average speed of 20.02 MPH. The system shows promising results and it can be enhanced to decrease the average error and to be tested with different types of vehicles and trucks.

CHAPTER 11

CONCLUSION AND FUTURE WORK

The wide spread of smartphones and the rich set of sensors that come with modern smartphones have encouraged developers to build intelligent transportation applications. Developers face difficulties to filter, fuse, and process sensors data. This motivated us to create SenSys framework that can process, filter, analyze, and understand sensors' readings in addition to the ability to extract vehicle dynamics. Many transportation applications can be built on top of the SenSys framework. SenSys was built to accommodate new sensors, filters, algorithms, and updates.

The existing ITS systems face many problems that need to be addressed, for example, several data collection technologies like surveillance cameras can track users and raise privacy concerns among drivers. Also, the cost of the intelligent transportation system varies from technology to another. Most existing technologies require the installation of sensors and devices in roads, the equipment could be too expensive in some cases. Some existing technologies like road detectors suffer from poor reliability due to improper installation. Many detectors like inductive loops need to be installed during road construction; the installation requires a saw cut to the pavement and stopping the traffic during the time of installation and maintenance. Because of the cost and the installation difficulty of installing new ITS technologies, the authorities install these technologies in few selected places. The data collected from different technologies are limited to the type of the sensor used in that technology, for example, the data collected by inductive loop detectors are vehicle passage, presence, count, and occupancy and the data collected by cameras are limited to footage, plate numbers, traffic density, and speed detection.

Traffic challenges are encouraging researchers to look for new technologies that can assist or replace the current systems. The high technology used in smartphones and their rich features in terms of processors, sensors, communication, and memory encouraged the researchers to utilize them in enhancing the existing ITS systems. The large spread of smartphones made it easy to deploy new low-cost technologies to collect data from drivers and vehicles. Current smartphones come with different types of sensors like, inertial sensors, camera, microphone, and GPS. The data derived

from these sensors encourage researchers to develop different types of applications in many domains like sports, health, localization, smart homes, and ITS.

The existing smartphones based ITS applications use phone either for data collection or to do one specific application. Developers face difficulties in developing smartphone ITS applications because of the difficulty in processing the noisy data produced by phones inertial sensors. More advanced smartphone-based ITS applications can be developed if there is a framework that does all the data processing and the feature extraction. This motivated us to build SenSys as a reliable smartphone based ITS framework that reads, processes, filters the raw data and extracts different features that can be used by developers to easily build different types of ITS applications.

Using SenSys set of APIs, we have built four ITS applications and included them as part of the SenSys library. The four applications are the InLanCom app which enables a vehicle to identify other vehicles in the same lane and exchange messages with them, the in-lane communication system proposed a communication scheme between vehicles in the same lane. On-board smartphones are used to detect the current lane and inertial sensors are used to generate a signature for the current lane. The signature is appended to the outgoing messages. Receivers decode the signature and decide whether to accept or deny the message with 95% accuracy. Experimental evaluation depicted how to extract lane's features (physical, and traffic), generate the signature, send it, and compare it at the receiver's side. *inLaneCom* works with different types and sizes of cars. It also works on different types of roads, and showed the best performance in city roads. The scecond application is the ParkZoom app which enables a vehicle to identify its exact parking spot in a parking lot. To accurately determine a vehicle's parking space using INS only, we used the parking lot infrastructure to overcome the acceleration noise problem. Turn detection locates the cars on turns, while error-corrected distance calculation and matching with the parking geometry allows ParkZoom to identify the parking spot. Real-world experiments with data collected from two simple and regular parking geometries showed good accuracy in detecting turns and parking-space determination. The third application is the GoGreen app which enables the driver to monitor the fuel consumption and CO_2 emission of his car, we developed an Android app that uses SenSys library to get the speed of the vehicle using inertial sensors, GPS, and OBD and show the average fuel consumption in liter per minute to the driver. The fourth

application is the vehicle type detection app which detects the type of the car based on the FHWA classification. The automatic vehicle type detection is an example of traffic applications that can use SenSys. The proposed automatic vehicle type detection uses smartphone inertial sensors to detect the number of axles, and then calculates the distances between them to determine vehicle's class and type.

Now, the ITS researchers and developers who need to build a smartphone applications can use SenSys framework to read, filter, fuse, and analyze sensors' readings. SenSys provides them with a set of APIs that enable their applications to understand vehcile's movements. The APIs can be easly called which will enable new ideas and applications in the ITS field.

In summary, this thesis contributes the following to the field of ITS: the design and development of SenSys framework that can be used by smartphone developers to build ITS applications since there is no such framework available today. SenSys uses newly developed algorithms to filter and fuse different sensors to correct the faulty readings and extract informative data. SenSys uses a new method to align smartphone's coordinates with vehicle's coordinates. SenSys provides a set of APIs that can be used by smartphone developers to build ITS applications. SenSys framework was used to design and develop a new and accurate method to identify a vehicle's parking spot using smartphone's inertial sensors only. In addition to that, SenSys framework was used to design and develop a new method that identifies vehicles in the same lane using smartphone's inertial sensors only, and a new method that detects vehicle's number of axles and class type using smartphone's inertial sensors only.

11.1 FUTURE WORK

Since SenSys is an expandable framework and important part of the future work is to expand the framework with more filters and methods to deal with different types of sensors. One of them is the microphone and the acoustic features extraction and processing. Another important issue we are planning to explore is to know if the phone is currently in use by the driver or not, and if the driver is using it for sending text messages, browsing, calling, or any other activity. If the phone is currently in use, can the algorithms still detects the vehicle dynamics like speed, turns, and stops? In addition, the inLaneCom system needs to find new ways to increase its accuracy on highways, there are few places that have a low accuracy because of the nature of

the traffic and roads. In addition to that, our future work includes expanding the system to establish a communication mean between fixed infrastructure and on-board smartphones, in-lane communication is needed in some places and intersections to enhance navigation, reduce traffic, and increase road safety. In addition to that, we need to explore more features and sensors to enhance highway lane identification accuracy. Since many applications can use the *inLaneCom* framework, part of our future work is to build more systems that use this framework to enhance road safety. The vehicle type application has limited its experiments to two types of vehicles. More types of vehicles can be tested and analyzed to have more accurate vehicle type detection SenSys can be expanded by using new sensors and algorithms. For example, there are different applications that use the camera like the lane departure warning applications, such applications use a rich set of image processing and computer vision algorithms to process the videos. Such algorithms could be added to the framework. The camera is part of the data collection layer, the image processing algorithms can be added to the data preparation layer and vehicle dynamics extraction layer.

REFERENCES

[1] US-DOT, "2015 transportation statistics annual report," 2015. [Online]. Available: http://www.rita.dot.gov/bts/sites/rita.dot.gov.bts/files/TSAR_2015_final_0.pdf

[2] T. Mehta, H. S. Mahmassani, and C. R. Bhat, "Methodologies for evaluating environmental benefits of intelligent transportation systems," Center for Transportation Research, Bureau of Engineering Research, University of Texas at Austin, Tech. Rep., 2001.

[3] H. A. Rakha, K. Ahn, K. Moran, B. Saerens, and E. Van den Bulck, "Virginia tech comprehensive power-based fuel consumption model: model development and testing," *Transportation Research Part D: Transport and Environment*, vol. 16, no. 7, pp. 492–503, 2011.

[4] FHWA, "FHWA vehicle classification definitions," 2015. [Online]. Available: https://www.fhwa.dot.gov/publications/research/infrastructure/pavements/ltpp/13091/002.cfm

[5] D. Schrank, B. Eisele, T. Lomax, and J. Bak, "2015 urban mobility scorecard," *Annual Urban Mobility Scorecard*, 2015.

[6] M. Dadafshar, "Accelerometer and gyroscopes sensors: operation, sensing, and applications," *Maxim Integrated [online]*, 2014.

[7] FHWA, "FHWA vehicle classification," 2015. [Online]. Available: http://onlinemanuals.txdot.gov/txdotmanuals/tri/vehicle_classification_using_fhwa_13category_scheme.htm

[8] C. Yang, D. McCollum, R. McCarthy, and W. Leighty, "Meeting an 80% reduction in greenhouse gas emissions from transportation by 2050: A case study in California," *Transportation Research Part D: Transport and Environment*, vol. 14, no. 3, pp. 147–156, 2009.

[9] M. Bell, "Environmental factors in intelligent transport systems," in *IEE Proceedings-Intelligent Transport Systems*, vol. 153, no. 2. IET, 2006, pp. 113–128.

[10] N. Lutsey and D. Sperling, "Greenhouse gas mitigation supply curve for the united states for transport versus other sectors," *Transportation Research Part D: Transport and Environment*, vol. 14, no. 3, pp. 222–229, 2009.

[11] T. D. Johnson, "Emergency vehicle notification system," July 8 2008, US Patent 7,397,356.

[12] J. W. Lee, "A machine vision system for lane-departure detection," *Computer vision and image understanding*, vol. 86, no. 1, pp. 52–78, 2002.

[13] I. De Vlieger, D. De Keukeleere, and J. Kretzschmar, "Environmental effects of driving behaviour and congestion related to passenger cars," *Atmospheric Environment*, vol. 34, no. 27, pp. 4649–4655, 2000.

[14] ATT, "AT&T quarterly earnings 4th quarter 2014," 2014. [Online]. Available: http://www.att.com/Investor/Earnings/4q14/slides_4q14.pdf

[15] Google, "Google play," 2014. [Online]. Available: https://support.google.com/googleplay

[16] Apple, "Quarterly earnings 3rd quarter 2014," 2014. [Online]. Available: http://www.apple.com/pr/library/2014/

[17] A. Alasaadi and T. Nadeem, "Unicoor: A smartphone unified coordinate system for its apps," in *2016 IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2016.

[18] I. Ustin, A. Alasaadi, M. Cetin, and T. Nadeem, "Detecting vehicle stops from smartphone accelerometer data," in *The 21st World Congress on Intellegent Transportation Systems, ITSWC2014*. ITSWC, 2014.

[19] A. Alasaadi, J. Aparicio, N. Tas, J. Rosca, and T. Nadeem, "Parkzoom: A parking spot identification system," in *Intelligent Transportation Systems - (ITSC), 2013 16th International IEEE Conference on*, Oct 2013, pp. 702–707.

[20] A. Alasaadi and T. Nadeem, "In-lane communication framework using smartphone's inertial sensors," in *Proceedings of the 20th annual international conference on Mobile computing and networking*. ACM, 2014, pp. 347–350.

[21] F. Amiri, E. Minge, and M. Culver, "Evaluation of non-intrusive technologies for traffic detection," *IMSA Journal*, vol. 39, no. 6, 2001.

[22] M. Dan, D. Jasek, and R. Parker, "Evaluation of some existing technologies for vehicle detection," *TX: Texas Transportation Institute, Report FHWA/TX-00/1715-S. College Station*, 1999.

[23] E. Minge, J. Kotzenmacher, and S. Peterson, "Evaluation of non-intrusive technologies for traffic detection," Minnesota Department of Transportation, Research Services Section, Tech. Rep., 2010.

[24] P. G. Michalopoulos, "Vehicle detection video through image processing: the autoscope system," *IEEE Transactions on vehicular technology*, vol. 40, no. 1, pp. 21–29, 1991.

[25] B. Coifman, D. Beymer, P. McLauchlan, and J. Malik, "A real-time computer vision system for vehicle tracking and traffic surveillance," *Transportation Research Part C: Emerging Technologies*, vol. 6, no. 4, pp. 271–288, 1998.

[26] P. G. Michalopoulos, R. D. Jacobson, C. A. Anderson, and T. B. DeBruycker, "Automatic incident detection through video image processing," *Traffic engineering & control*, vol. 34, no. 2, 1993.

[27] B. T. Morris and M. M. Trivedi, "Learning, modeling, and classification of vehicle track patterns from live video," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 3, pp. 425–437, 2008.

[28] G. Zhang, R. Avery, and Y. Wang, "Video-based vehicle detection and classification system for real-time traffic data collection using uncalibrated video cameras," *Transportation Research Record: Journal of the Transportation Research Board*, no. 1993, pp. 138–147, 2007.

[29] A. De la Escalera, J. M. Armingol, and M. Mata, "Traffic sign recognition and analysis for intelligent vehicles," *Image and vision computing*, vol. 21, no. 3, pp. 247–258, 2003.

[30] Z. Sun, G. Bebis, and R. Miller, "On-road vehicle detection: A review," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, no. 5, pp. 694–711, 2006.

[31] X. Zhu, Q. Li, and G. Chen, "Apt: Accurate outdoor pedestrian tracking with smartphones," in *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013, pp. 2508–2516.

[32] J. Yang, S. Sidhom, G. Chandrasekaran, T. Vu, H. Liu, N. Cecan, Y. Chen, M. Gruteser, and R. P. Martin, "Detecting driver phone use leveraging car speakers," in *Proceedings of the 17th annual international conference on Mobile computing and networking*. ACM, 2011, pp. 97–108.

[33] C.-W. You, M. Montes-de Oca, T. J. Bao, N. D. Lane, H. Lu, G. Cardone, L. Torresani, and A. T. Campbell, "Carsafe: a driver safety app that detects dangerous driving behavior using dual-cameras on smartphones," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. ACM, 2012, pp. 671–672.

[34] J. White, C. Thompson, H. Turner, B. Dougherty, and D. C. Schmidt, "Wreckwatch: Automatic traffic accident detection and notification with smartphones," *Mobile Networks and Applications*, vol. 16, no. 3, pp. 285–303, 2011.

[35] L. S. Cai and G. B. Ning, "Adaptive driving speed guiding to avoid red traffic lights," in *Applied Mechanics and Materials*, vol. 347. Trans Tech Publ, 2013, pp. 3832–3836.

[36] M. Fazeen, B. Gozick, R. Dantu, M. Bhukhiya, and M. C. González, "Safe driving using mobile phones," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 13, no. 3, pp. 1462–1468, 2012.

[37] M. Ruta, F. Scioscia, F. Gramegna, G. Loseto, and E. Di Sciascio, "Knowledge-based real-time car monitoring and driving assistance." in *SEBD*, 2012, pp. 289–294.

[38] S. Choi, J. Kim, D. Kwak, P. Angkititrakul, and J. H. Hansen, "Analysis and classification of driver behavior using in-vehicle can-bus information," in *Biennial Workshop on DSP for In-Vehicle and Mobile Systems*, 2007, pp. 17–19.

[39] A. K. Shaout and A. E. Bodenmiller, "A mobile application for monitoring inefficient and unsafe driving behaviour," 2011.

[40] V. C. Magaña and M. M. Organero, "Artemisa: Using an android device as an eco-driving assistant," 2011.

[41] OBDdoctor, "Obddoctor app," 2015. [Online]. Available: http://www.obdautodoctor.com

[42] Torque, "Torque app," 2015. [Online]. Available: http://torque-bhp.com

[43] M. Lan, M. Rofouei, S. Soatto, and M. Sarrafzadeh, "Smartldws: A robust and scalable lane departure warning system for the smartphones," in *Intelligent Transportation Systems, 2009. ITSC'09. 12th International IEEE Conference on*. IEEE, 2009, pp. 1–6.

[44] S. Singh, S. Nelakuditi, R. Roy Choudhury, and Y. Tong, "Your smartphone can watch the road and you: mobile assistant for inattentive drivers," in *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*. ACM, 2012, pp. 261–262.

[45] W. H. Ling and W. C. Seng, "Traffic sign recognition model on mobile device," in *Computers & Informatics (ISCI), 2011 IEEE Symposium on*. IEEE, 2011, pp. 267–272.

[46] M. Munoz-Organero and V. C. Magana, "Validating the impact on reducing fuel consumption by using an ecodriving assistant based on traffic sign detection and optimal deceleration patterns," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 14, no. 2, pp. 1023–1028, 2013.

[47] C. Thompson, J. White, B. Dougherty, A. Albright, and D. C. Schmidt, "Using smartphones to detect car accidents and provide situational awareness to emergency responders," in *Mobile Wireless Middleware, Operating Systems, and Applications*. Springer, 2010, pp. 29–42.

[48] M.-H. L. N. Condro, M.-H. Li, and R.-I. Chang, "Motosafe: Active safe system for digital forensics of motorcycle rider with android," *International Journal of Information and Electronics Engineering*, vol. 2, no. 4, pp. 612–616, 2012.

[49] L. Bedogni, M. Di Felice, and L. Bononi, "By train or by car? detecting the user's motion type through smartphone sensors data," in *Wireless Days (WD), 2012 IFIP*, Nov 2012, pp. 1–6.

[50] S. Hemminki, P. Nurmi, and S. Tarkoma, "Accelerometer-based transportation mode detection on smartphones," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2013, p. 13.

[51] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, 2008, pp. 323–336.

[52] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan, "The pothole patrol: using a mobile sensor network for road surface monitoring," in *Proceedings of the 6th international conference on Mobile systems, applications, and services*. ACM, 2008, pp. 29–39.

[53] J. Karuppuswamy, V. Selvaraj, M. M. Ganesh, and E. L. Hall, "Detection and avoidance of simulated potholes in autonomous vehicle navigation in an unstructured environment," in *Intelligent Systems and Smart Manufacturing*. International Society for Optics and Photonics, 2000, pp. 70–80.

[54] G. Strazdins, A. Mednis, G. Kanonirs, R. Zviedris, and L. Selavo, "Towards vehicular sensor networks with android smartphones for road surface monitoring," in *2nd International Workshop on Networks of Cooperating Objects (CONET11), Electronic Proceedings of CPS Week*, vol. 11, 2011.

[55] H. Eren, S. Makinist, E. Akin, and A. Yilmaz, "Estimating driving behavior by a smartphone," in *Intelligent Vehicles Symposium (IV), 2012 IEEE*. IEEE, 2012, pp. 234–239.

[56] R. G. Aldunate, O. A. Herrera, and J. P. Cordero, "Early vehicle accident detection and notification based on smartphone technology," in *Ubiquitous Computing and Ambient Intelligence. Context-Awareness and Context-Driven Interaction*. Springer, 2014.

[57] J. Almazán, L. M. Bergasa, J. J. Yebes, R. Barea, and R. Arroyo, "Full auto-calibration of a smartphone on board a vehicle using imu and gps embedded sensors," in *Intelligent Vehicles Symposium (IV), 2013 IEEE*. IEEE, 2013, pp. 1374–1380.

[58] H. Aly, A. Basalamah, and M. Youssef, "Lanequest: An accurate and energy-efficient lane detection system," *arXiv preprint arXiv:1502.03038*, 2015.

[59] K. Kunze, P. Lukowicz, K. Partridge, and B. Begole, "Which way am I facing: Inferring horizontal device orientation from an accelerometer signal," in *Wearable Computers, 2009. ISWC'09. International Symposium on.* IEEE, 2009, pp. 149–150.

[60] D. Pai, M. Malpani, I. Sasi, N. Aggarwal, and P. Mantripragada, "Padati: A robust pedestrian dead reckoning system on smartphones," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on.* IEEE, 2012, pp. 2000–2007.

[61] J. Morales, D. Akopian, and S. Agaian, "Human activity recognition by smartphones regardless of device orientation," in *IS&T/SPIE Electronic Imaging.* International Society for Optics and Photonics, 2014, pp. 90 300I–90 300I.

[62] Y. Wang, J. Yang, H. Liu, Y. Chen, M. Gruteser, and R. P. Martin, "Sensing vehicle dynamics for determining driver phone use," in *Proceeding of the 11th annual international conference on Mobile systems, applications, and services.* ACM, 2013, pp. 41–54.

[63] J. Dai, J. Teng, X. Bai, Z. Shen, and D. Xuan, "Mobile phone based drunk driving detection," in *Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2010 4th International Conference on-NO PERMISSIONS.* IEEE, 2010, pp. 1–8.

[64] Y. Zhao, Y. Zhang, T. Yu, T. Liu, X. Wang, X. Tian, and X. Liu, "Citydrive: A map-generating and speed-optimizing driving system," in *INFOCOM, 2014 Proceedings IEEE.* IEEE, 2014, pp. 1986–1994.

[65] H. Han, J. Yu, H. Zhu, Y. Chen, J. Yang, Y. Zhu, G. Xue, and M. Li, "Senspeed: Sensing driving conditions to estimate vehicle speed in urban environments," in *INFOCOM, 2014 Proceedings IEEE.* IEEE, 2014, pp. 727–735.

[66] eLibera. Find my car app. [Online]. Available: https://play.google.com/store/apps/details?id=com.elibera.android.findmycar

[67] J. Charles. Mycar locator app. [Online]. Available: https://play.google.com/store/apps/details?id=com.nomadrobot.mycarlocatorfree

[68] J. Werb and C. Lanzl, "Designing a positioning system for finding things and people indoors," *Spectrum, IEEE*, vol. 35, no. 9, pp. 71–78, 1998.

[69] N. Patwari, A. O. Hero III, M. Perkins, N. S. Correal, and R. J. O'dea, "Relative location estimation in wireless sensor networks," *Signal Processing, IEEE Transactions on*, vol. 51, no. 8, pp. 2137–2148, 2003.

[70] H. H. Liu and G. K. Pang, "Accelerometer for mobile robot positioning," *Industry Applications, IEEE Transactions on*, vol. 37, no. 3, pp. 812–819, 2001.

[71] M. Reinstein and M. Hoffmann, "Dead reckoning in a dynamic quadruped robot: Inertial navigation system aided by a legged odometer," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on.* IEEE, 2011, pp. 617–624.

[72] P. Davidson, J. Hautamäki, J. Collin, and J. Takala, "Improved vehicle positioning in urban environment through integration of gps and low-cost inertial sensors," in *Proceedings of the the European Navigation Conference (ENC09)*, 2009.

[73] J. Georgy, A. Noureldin, M. J. Korenberg, and M. M. Bayoumi, "Low-cost three-dimensional navigation solution for riss/gps integration using mixture particle filter," *Vehicular Technology, IEEE Transactions on*, vol. 59, no. 2, pp. 599–615, 2010.

[74] L. I. Iozan, J. Collin, and J. Takala, "Integrating mems sensors with gps technology for obtaining a continuous navigation solution in urban areas," 2011.

[75] L. Zhao, W. Y. Ochieng, M. A. Quddus, and R. B. Noland, "An extended kalman filter algorithm for integrating gps and low cost dead reckoning system data for vehicle performance and emissions monitoring," *Journal of Navigation*, vol. 56, no. 2, pp. 257–275, 2003.

[76] V. Tyagi, S. Kalyanaraman, and R. Krishnapuram, "Vehicular traffic density state estimation based on cumulative road acoustics," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 3, pp. 1156–1166, 2012.

[77] R. N. Jazar, "Quarter car," in *Vehicle Dynamics: Theory and Application*. Springer, 2008, pp. 931–975.

[78] J. Wang and R. G. Longoria, "Coordinated and reconfigurable vehicle dynamics control," *Control Systems Technology, IEEE Transactions on*, vol. 17, no. 3, pp. 723–732, 2009.

[79] J. He, D. Crolla, M. Levesley, and W. Manning, "Coordination of active steering, driveline, and braking for integrated vehicle dynamics control," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 220, no. 10, pp. 1401–1420, 2006.

[80] G. Casiez, N. Roussel, and D. Vogel, "1 euro filter: A simple speed-based low-pass filter for noisy input in interactive systems," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '12. New York, NY, USA: ACM, 2012, pp. 2527–2530. [Online]. Available: http://doi.acm.org/10.1145/2207676.2208639

[81] P. Lawitzki, "Application of dynamic binaural signals in acoustic games," *Stuttgart Media University*, 2012.

[82] A. Mohamed and K. Schwarz, "Adaptive kalman filtering for INS/GPS," *Journal of geodesy*, vol. 73, no. 4, pp. 193–203, 1999.

[83] J. C. Herrera, D. B. Work, R. Herring, X. J. Ban, Q. Jacobson, and A. M. Bayen, "Evaluation of traffic data obtained via gps-enabled mobile phones: The mobile century field experiment," *Transportation Research Part C: Emerging Technologies*, vol. 18, no. 4, pp. 568–583, 2010.

[84] V. Douangphachanh and H. Oneyama, "A study on the use of smartphones for road roughness condition estimation," *Journal of the Eastern Asia Society for Transportation Studies*, vol. 10, no. 0, pp. 1551–1564, 2013.

[85] M. Müller, "Dynamic time warping," *Information retrieval for music and motion*, pp. 69–84, 2007.

[86] M. G. Wing, A. Eklund, and L. D. Kellogg, "Consumer-grade global positioning system (gps) accuracy and reliability," *Journal of Forestry*, vol. 103, no. 4, pp. 169–173, 2005.

[87] C. Hunter. Green meter app. [Online]. Available: http://www.greenmeter.com/

[88] A. King, "Inertial navigation-past, present, and future," in *Airborne Navigation Systems Workshop (Digest No. 1997/169), IEE Colloquium on.* IET, 1997, pp. 3–1.

[89] S. Boonmee and P. Tangamchit, "Portable reckless driving detection system," in *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2009. ECTI-CON 2009. 6th International Conference on*, vol. 1. IEEE, 2009, pp. 412–415.

[90] P. Aksamit and M. Szmechta, "Distributed, mobile, social system for road surface defects detection," in *Computational Intelligence and Intelligent Informatics (ISCIII), 2011 5th International Symposium on.* IEEE, 2011, pp. 37–40.

[91] A. Mednis, G. Strazdins, R. Zviedris, G. Kanonirs, and L. Selavo, "Real time pothole detection using android smartphones with accelerometers," in *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on.* IEEE, 2011, pp. 1–6.

[92] H. Claussen, J. Aparicio, J. Rosca, and N. C. Tas, "Ipass: Intelligent pavement signaling system," in *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on.* IEEE, 2012, pp. 666–671.

[93] Y. Wang, N. Dahnoun, and A. Achim, "A novel system for robust lane detection and tracking," *Signal Processing*, vol. 92, no. 2, pp. 319–334, 2012.

[94] R. A. De Lorenzo and M. A. Eilers, "Lights and siren: A review of emergency vehicle warning systems," *Annals of emergency medicine*, vol. 20, no. 12, pp. 1331–1335, 1991.

[95] C. Q. Howard, A. J. Maddern, and E. P. Privopoulos, "Acoustic characteristics for effective ambulance sirens," *Acoustics Australia*, vol. 39, no. 2, pp. 43–53, 2011.

[96] V. Gradinescu, C. Gorgorin, R. Diaconescu, V. Cristea, and L. Iftode, "Adaptive traffic lights using car-to-car communication," in *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th.* IEEE, 2007, pp. 21–25.

[97] A. Hesham, A. Abdel-Hamid, and M. A. El-Nasr, "A dynamic key distribution protocol for pki-based vanets," in *Wireless Days (WD), 2011 IFIP*. IEEE, pp. 1–3.

[98] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full sha-1," in *Advances in Cryptology–CRYPTO 2005*. Springer, 2005, pp. 17–36.

[99] V. Turri, A. Carvalho, H. E. Tseng, K. H. Johansson, and F. Borrelli, "Linear model predictive control for lane keeping and obstacle avoidance on low curvature roads," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE, 2013, pp. 378–383.

[100] K.-Y. Chiu and S.-F. Lin, "Lane detection using color-based segmentation," in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.* IEEE, 2005, pp. 706–711.

[101] C. Shun-Hung, C.-W. Hsu, and Y. Po-Kai, "Lane curvature detection system by utilizing vehicular and inertial sensing signals," Feb. 17 2015, uS Patent 8,958,925.

[102] K. Ahn, H. Rakha, A. Trani, and M. Van Aerde, "Estimating vehicle fuel consumption and emissions based on instantaneous speed and acceleration levels," *Journal of transportation engineering*, vol. 128, no. 2, pp. 182–190, 2002.

[103] R. Akcelik and M. Besley, "Operating cost, fuel consumption, and emission models in aasidra and aamotion," in *25th Conference of Australian Institutes of Transport Research (CAITR 2003)*, 2003, pp. 1–15.

[104] J. Van Mierlo, G. Maggetto, E. Van de Burgwal, and R. Gense, "Driving style and traffic measures-influence on vehicle emissions and fuel consumption," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 218, no. 1, pp. 43–50, 2004.

[105] A. Alessandrini, A. Cattivera, F. Filippi, and F. Ortenzi, "Driving style influence on car co2 emissions," in *2012 International Emission Inventory Conference*, 2012.

[106] I. E. EUROPE, "European campaign on improving driving behaviour, energy-efficiency and traffic safety (ecodriven)," 2015. [Online]. Available: https://ec.europa.eu/energy/intelligent/projects/en/projects/ecodriven

VITA

Abdulla Ahmed Alasaadi
Department of Computer Science
Old Dominion University
Norfolk, VA 23529

EDUCATION:

2003: BSc in Copmuter Science, University of Bahrain, Bahrain.

2005: MSc in Advanced Computer Science, Lancaster University, Lancaster, England.

2007: Postgraduate Certificate in Academic Practice PCAP,York St John University, York, England.