# Seamless Interactions Between Humans and Mobility Systems

by

Dongyao Chen

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2020

Doctoral Committee:

Professor Kang G. Shin, Chair
Associate Professor Chad Jenkins
Professor Huei Peng
Associate Professor Alanson Sample

Dongyao Chen

chendy@umich.edu

ORCID iD: 0000-0002-5223-7304

*To my wife and my parents*

*For their unconditional love and support*

*And to Anxin*

*Your arrival is the best gift we could ever have*

# ACKNOWLEDGEMENTS

Words can not express my gratitude enough for my advisor, Kang G. Shin. His patience and guidance not only grant me the freedom to follow my passion in research but also guide me to become an independent thinker. Prof. Shin exceeds all my expectations for an advisor from all perspectives — his style of mentorship is the true embodiment of how a Jedi Master would nurture and train his fellow apprentices.

I would like to thank my committee members Alanson Sample, Chad Jenkins, and Huei Peng for their constructive questions and comments that help improve this dissertation. I am grateful to Kyu-Han Kim and Yurong Jiang for their mentorship at the HP Labs — I remember these inspiring conversations with them on making impactful research works that also scale in the real-world. I thank my undergraduate advisor, Xinbing Wang, for encouraging me to pursue excellence.

I am fortunate to have the companion of all the fellow RTCLers during my PhD journey. In particular, I would like to thank Kyong-Tak Cho, together we have had tackled many uncharted challenges not only in conducting research projects but also in sketching future blueprints. I would like to thank Mert, Arun, Liang, Chun-Yu, and Youngmoon: your commitment and enthusiasm for life and work are contagious. I thank Huan, Yu-Chih, Xiaoen, and Sihui for our brainstorming sessions and helping me fit in our lab. On my first day, Prof. Shin assigned me a seat and told me "Dongyao, when you are confused, perhaps you can get inspired by the people around you." For that, I would like to thank Taeju, Kassem, Juncheng, Karen, Tim, Hsun-Wei, Youssef, Eric, Yuanchao, Kyusuk, Suining, Xiufeng, HoonSung, Haichuan, Jinkyu, DaeHan, Sunhyun, Zhao, Xiaoyan, Chaocan, and

Jiaqing. I couldn’t even imagine having this journey without the collective wisdom and warmth from the big RTCL family.

I would like to thank my father Jiansheng Chen, my mother Cuilian Zhang, my parents-in-law, and my grandfather Zhenqing Zhang for their unconditional love and support. My parents have been my role models for demonstrating the level of happiness one could achieve by persistently pursuing his/her dream career. I would like to thank my wife, Sai Zhang. She crossed half the globe from Singapore to Ann Arbor to build our own family and pursue her career goals. Her kindness and trust have been supporting me to overcome numerous hardships for the past few years. For that, I am eternally grateful, and I can’t wait to start our next endeavor.

# TABLE OF CONTENTS

# LIST OF FIGURES

**Figure**

# LIST OF TABLES

**Table**

# ABSTRACT

As mobility systems, including vehicles and roadside infrastructure, enter a period of rapid and profound change, it is important to enhance interactions between people and mobility systems. Seamless human—mobility system interactions can promote widespread deployment of engaging applications, which are crucial for driving safety and efficiency.

The ever-increasing penetration rate of ubiquitous computing devices, such as smartphones and wearable devices, can facilitate realization of this goal. Although researchers and developers have attempted to adapt ubiquitous sensors for mobility applications (e.g., navigation apps), these solutions often suffer from limited usability and can be risk-prone. The root causes of these limitations include the low sensing modality and limited computational power available in ubiquitous computing devices.

We address these challenges by developing and demonstrating that novel sensing techniques and machine learning can be applied to extract essential, safety-critical information from drivers natural driving behavior, even actions as subtle as steering maneuvers (e.g., left-/righthand turns and lane changes). We first show how ubiquitous sensors can be used to detect steering maneuvers regardless of disturbances to sensing devices. Next, by focusing on turning maneuvers, we characterize drivers driving patterns using a quantifiable metric. Then, we demonstrate how microscopic analyses of crowdsourced ubiquitous sensory data can be used to infer critical macroscopic contextual information, such as risks present at road intersections. Finally, we use ubiquitous sensors to profile a driver's behavioral patterns on a large scale; such sensors are found to be essential to the analysis and improvement of drivers driving behavior.

# CHAPTER I

# Introduction

Many novel mobility technologies have been proposed in recent years. For example, newer high-end vehicles are equipped with dedicated sensing modules (e.g., cameras and radar) to realize advanced driving assistance systems (ADAS) that include several safety features (e.g., lane departure warning [95] and facial-based drowsy driver detection [39]) to enhance driving safety. High-definition roadmap service providers, including Google StreetView [9], TomTom [42], and Here [20], have recently deployed specialized sensor-rich mapping vehicles to collect roadside images and build a comprehensive dataset to provide more accurate car-based navigation services.

Although existing technologies have proven effective in enhancing the safety and efficiency of on-road transportation, they are often *expensive*. Associated costs contribute to a bottleneck that hinders widespread application of these advances. For example, many of the latest safety features are available for purchase as additional bundles rather than being included in the basic price of a car. This bottleneck will presumably continue to narrow as mobility technology becomes more sophisticated. Approximately 1,400 [34] self-driving cars are being *tested* on U.S. roads; by comparison, the number of *operational* legacy cars in the U.S. (i.e., those with an automation level of 2 or lower based on the SAE J3016 [55] standard) has reached 276 million [32].

The disproportionate ratio between *smart* and legacy mobility systems is problematic for two reasons.

- **Protecting individuals well-being on the road.** Democratizing advanced technology cross-mobility systems would decrease accidents, thus improving road safety and efficiency. For example, the anti-lock braking system (ABS), which prevents wheels from locking up while braking, was introduced in the 1970s [4] as a safety feature exclusive to high-end cars given its high manufacturing costs. Real-world data and statistics have confirmed the effectiveness of ABS in decreasing fatal accidents: the collision rate has "fallen substantially" by at least 6% for passenger cars according to a National Highway Traffic Safety Administration study (Chapter 3 of Study DOT-HS-811-182 [137]). In the United States, however, ABS did not become a popular feature (i.e., more than 50% of new passenger cars were equipped with the system) until 1994 (Chapter 1.3 of [137]). Thanks to a notable decline (Chapter 4.4 of [137]) in manufacturing costs, ABS only became a mandatory feature of all new cars in the United States beginning in 2013 — more than four decades after its debut.
- **Discovering insights.** A large number of users/participants are often necessary to reveal accurate real-world insights. For instance, Google Maps live traffic alerts [24], which highlight congested road areas in red, are derived from travel-speed data (from smartphones GPS readings) from various app users. Collecting data from a small number of users could return inaccurate results; users who upload data may be traveling at particularly fast/slow speeds, rendering the collected data unrepresentative of actual driving conditions.

A "silver bullet" to drastically lower the costs of existing mobility technologies may not yet exist; therefore, we focus on democratizing novel mobility technologies by using more accessible sensing platforms and new data analytics schemes. Our goal is to develop practical systems that can seamlessly facilitate human users interactions with mobility systems.

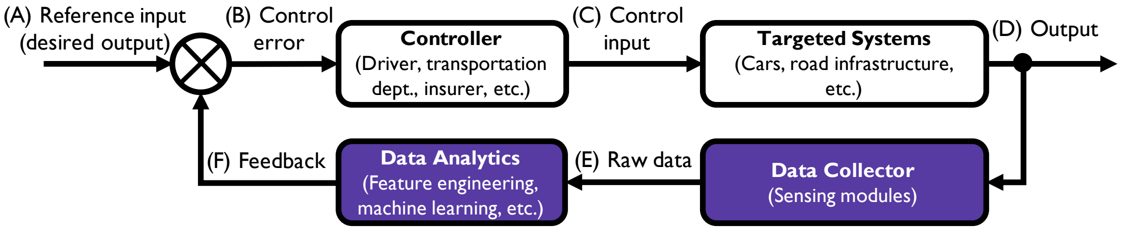


Figure 1.1: Block diagram of a feedback control system for analyzing the interaction between transportation stakeholders (e.g., human users) and mobility systems.

To *systematically* explore why seamless *interaction* may be especially advantageous in mobility ecosystems, we have adapted the concept of classic closed-loop feedback control. As noted by Karl Astrom [96], one of the early pioneers in control theory, the "magic" of this feedback control system is rooted in fashioning working systems from components that otherwise perform poorly. Fig. 1.1 depicts a control system in which the blocks and arrows denote components and signals, respectively. Based on classic feedback control, reference input A is often the desired output; control error B represents the difference between the reference input and measured output. The control error is sent to the controller to generate control input C for the targeted system (e.g., a mobility system). The data collector can then collect/measure system output D. Generated raw data E is next fed into the data analytics module to generate feedback F. The objective of a feedback control system is to easily reduce the difference between output and reference input, namely through fast-converging, minimal oscillations. This concept can be applied to complex real-world systems [93]. The goal of navigation apps, such as Google Maps and Waze, is to help drivers arrive at their destinations in a timely manner. Based on a drivers current location and planned destination (desired output A), these apps first derive a path and provide guidance (control error B) to the driver (controller), who operates (control input C) the car (targeted system). Given the frequency with which users follow the wrong path (e.g., miss a highway exit), navigation apps must continuously monitor a drivers current location (i.e., system output D), which the path-planning algorithm needs in order to update the path (i.e., feedback F) for the driver. After the driver arrives at their destination, navigation is terminated as guidance is no longer needed (i.e., the system has minimal control error).

A seamless interaction channel is pivotal to this process; the channel harvests the feedback loop (purple components in Fig. 1.1) using novel mobility technologies to (a) recognize real-world disturbances in a *timely* manner and (b) enable a non-obstructive channel to feed useful information back to the controller, thus facilitating the desired goal. In this thesis, we refer to our proposed technologies as *seamless human—mobility interaction* (HMI).

Herein, we propose systems that can seamlessly facilitate HMI by harvesting ubiquitous computing devices (e.g., smartphones and wearables) [171] that possess three key features: (1) low costs; (2) a pair-wise connection and Internet connection; and (3) the ability to support versatile applications via software development. Although researchers have attempted to adapt ubiquitous sensors for mobility applications, such as navigation apps, these efforts are often plagued by limited usability and may be risk-prone. As we explain in Sec. 1.2, the root causes of these limitations include the limited sensing modality and computational power supported by ubiquitous computing devices.

Accordingly, the statement of this thesis is twofold:

> *(a) explore the methodology to effectively harvest ubiquitous sensing and realize seamless HMI and (b) discover novel applications from real-world mobility systems that can be enabled/improved by our proposed technologies.*

As described in Sec. 1.3, seamless HMI enabled by ubiquitous sensing is *unlikely* to be a feasible standalone platform (e.g., to replace vehicle cameras, radar, and so forth) for all mobility applications. Rather, we will demonstrate how our proposed approaches can serve as the primary sensing interface for certain tasks; for other tasks, our methods represent secondary/complementary sensing modalities to promote the robustness and versatility of relevant applications.

We summarize the general background and constraints of ubiquitous sensing in

Sec. 1.1 and 1.2, respectively. In Sec. 1.3, we discuss the promise and challenges of the state-of-the-art ubiquitous sensing techniques in achieving seamless HMI. We summarize the contributions of this thesis in Sec. 1.4, and Sec. 1.5 outlines the structure of the remainder of the thesis.

## 1.1 Background in Ubiquitous Sensing

Ubiquitous sensing [171] represents a category of accessible sensing technologies that leverage sensory data from low-cost computing platforms. This area is experiencing rapid technological and commercial growth due to the quickly evolving capabilities of mobile devices (e.g., smartphones, wearable devices, and the Internet of Things) as such devices shift from pure communication devices (e.g., cell phones) to *smart* devices. Mobile devices are practical platforms for enabling seamless HMI at scale thanks to their pervasiveness, versatility, and connectivity.

### 1.1.1 Pervasiveness

The relatively low costs associated with mobile devices have led to worldwide deployment and popularity, providing a significant advantage over other computational platforms. According to recent surveys [35, 40], consumers collectively owned more than 2.1 billion smartphones in 2016; the ownership rate of smartphones in developed areas is well over 87%, and proportions are expanding rapidly in developing regions of Asia and Africa, having increased from a median of 21% in 2013 to 37% in 2015.

The immense popularity of mobile platforms presents a unique opportunity for achieving large-scale sensing. Mobile devices can easily cover remote and/or developing regions. Furthermore, they can collect data as many times as needed within a given area of interest (e.g., a road segment) selected by the data collection entity, which is essential for building a comprehensive dataset.

### 1.1.2 Versatility

Mobile devices are capable of various lightweight tasks (i.e., those requiring less energy and computational overhead than larger tasks) thanks to embedded sensors such as cameras and motion sensors. For example, dual-camera systems have become the default configuration to capture detailed image data via smartphone. An inertial measurement unit (IMU), a type of highly integrated micro-electro-mechanical system (MEMS) chip, is also used to accurately capture the motion of an ego-mobile device with respect to angular speed, acceleration, and magnetic fields via a gyroscope, accelerometer, and magnetometer, respectively.

### 1.1.3 Connectivity

Mobile devices have flexible Internet connectivity; owners of smartphones and/or smartwatches [8] can use either cellular or WLAN (e.g., WiFi) networks to access the Internet. This flexibility endows mobile devices with unique advantages over other sensing platforms for collaborative mobility applications.

**Internet Access.** Internet access enables acquisition of information beyond the ego-cars perception range. For example, when a weather application broadcasts a tornado alert in a certain area, smartphone navigation apps can gather and process this information to plan alternate routes and minimize the effects of weather on drivers travel.

**Crowdsensing.** Crowdsensing [119] refers to crowdsourcing sensor data from numerous sensory devices. Crowdsensing has two key advantages: (1) rapid data collection when the user base is large and (2) greater diversity of data. For mobility applications that use mobile crowdsensing, each data collector is a mobile device that resides at, or is carried by an on-road entity (e.g., a passenger car). Sensor data can thus be gathered from the crowd to help construct a large-scale dataset, which is a valuable resource for data mining and machine learning tasks.

## 1.2 Limitations of Ubiquitous Sensing

Although mobile devices can play versatile roles in various sensing tasks, two fundamental obstacles complicate their adoption for mobility scenarios.

### 1.2.1 Energy Capacity

Most mobile devices have limited energy capacity because they were originally designed and manufactured for lightweight tasks (e.g., phone calls, web browsing, and gaming). As discussed throughout the ensuing subsections, limited battery life is a primary bottleneck to mobile devices hardware and software design.

### 1.2.2 Sensing Modality and Fidelity

Due to limited energy capacity and manufacturing costs, few sensor modalities are currently supported across mobile devices. For instance, to detect driving motion (e.g., acceleration or braking), devices with access to an in-vehicle network (i.e., controller-area network [27]) can easily incorporate at least 18 types of data, including throttling, gas/brake pedal position, and tire pressure. In contrast, mobile devices only support two types of sensory data (i.e., from a gyroscope and accelerometer) to capture vehicle movement.

To lower manufacturing costs, mobile sensors tend to be built with relatively lower-quality components than other sensing platforms (e.g., in-vehicle embedded modules). The most popular mobile IMU sensor chip, Bosch BMI160 [7] — available in the Pixel XL, Pixel 2, iPhone 8, and iPhone Xhas a unit price of less than $2 and a zero-drifting problem caused by operating temperature. In self-driving cars, IMU sensor data are generated through fiberoptic IMU modules, which exhibit much less zero drifting but cost more than $12,000; this price is prohibitive for large-scale production. Although embedded sensors are expected to continue to improve (e.g., BMI160s accuracy is 40% better than its predecessors), overcoming imperfect data (e.g., noise) from error-prone hardware continues to hinder the integration of mobile devices into smart transportation application(s).

### 1.2.3 Diversity of Device Configurations

The rapid growth of mobile devices has yielded an array of hardware and/or software configurations. Such product diversity may create compatibility issues. From the hardware perspective, different device models may contain distinct sensory modules, which can cause sensory data to vary in quality (e.g., sample rate or zero-drift error). A recent market survey [3] showed that Android devices were affiliated with more than 1,000 brands and 24,000 device models as of 2015. On the software side, operating systems (OSs) and software versions may also generate inconsistent sensory data across platforms.

## 1.3 State of the Art

The prospect of using mobile devices in mobility applications has attracted widespread attention from academia and industry. Industry professionals are intrigued by the business potential of integrating mobile devices into smart transportation technology. Despite being in its infancy, the topic has also piqued the interest of auto insurance companies, ADAS developers, and navigation service providers. Research efforts have revolved around the development of new application models by leveraging mobile devices sensing capabilities. In what follows, we have categorized such efforts by the type of sensory data used.

### 1.3.1 Image Data

Camera-based mobile apps are powered by the enhanced camera systems and computational power of mobile devices. ADAS developers (e.g., the Drivea app [12], available on Google Play and the App Store) have introduced mobile apps that use smartphone cameras to enhance driving safety. Specifically, if a smartphone is mounted in a car (e.g., on the windshield with a mount), the camera-captured video stream can warn the driver of tailgating if the distance between the front car and ego-car is too close. In a related vein, the research project CarSafe [178] uses an embedded dual-camera system to monitor activity in front of the car along with the drivers facial expression to detect drowsy driving. However, performing computer vision tasks locally on mobile platforms can incur considerable overhead. The strictly enforced privacy protection on smartphones also means that if a mobile OS detects camera usage, then other running apps will be interrupted by the display of a captured image/video on-screen; this interruption can severely undermine the apps usability.

### 1.3.2 Location Data

Location data (e.g., GPS data) consist of geolocation information collected while a user is driving. Navigation services use this information to infer the degree of traffic congestion by calculating drivers average moving speed based on a series of location data. A recent study [172] indicated the feasibility of constructing a users driving-style profile (i.e., acceleration and braking behavior) on the basis of long-term (i.e., at least 2 months worth) location data.

### 1.3.3 Motion Data.

Motion data refers to time-series IMU sensor data that capture an ego-smartphones movement; the gyroscope and accelerometer in an IMU sensor can capture the vehicles steering and acceleration/braking maneuvers if the smartphones positioning is fixed with respect to the car. For example, some auto insurance companies [57, 61] allow users to enroll in a usage-based insurance (UBI) program: drivers install a dedicated smartphone app, after which insurance companies can adjust users premiums and/or deductibles based on their driving performance.

### 1.3.4 Related Work

A technological ecosystem capable of facilitating a mobility ecosystem with ubiquitous sensing is still in its infancy. The extant literature has identified two main issues in this regard.

**Limited usability.** Many studies have fallen short in terms of using mobile devices in a versatile and non-obtrusive manner. IMU sensor data on mobile devices appear severely underutilized in mobility applications; at present, such data have mainly been explored for lightweight tasks given the limited contextual information IMU sensors can provide compared to other types of sensory data (e.g., images). In fact, IMU data can only be used to detect abrupt braking and acceleration behavior based on statistical patterns of time-series accelerometer data. More sophisticated tasks, such as determining a drivers identity, were originally thought to *only* be achievable with context-rich sensor data. For instance, Uber and Lyft ask drivers to take weekly selfies as a way to detect fraudulent drivers and/or criminals. Yet this strategy has technical limitations related to photo quality (e.g., insufficient lighting when taking a picture). This method may also fail in practice because it is not continuous; a "fake" driver could impersonate a legitimate driver between two check-ins.

**Risk-prone.** Most mobility applications on mobile devices cannot yet incorporate safety features for two reasons: (1) the sensing modalities of mobile devices are limited, and (2) the safety information is often implicit. For example, legacy navigation apps such as Google Maps have been using the estimated time of arrival as the sole route-planning criterion; however, this mechanism may generate risky routes (e.g., consecutive unprotected left turns). Such routes could be risky for drivers, especially novices. Recently, navigation apps such as Waze have begun to display road-context information, such as construction and speed traps, based on users manual map annotations. Although these types of supplementary information might be useful to drivers, this usage paradigm remains to fall short in terms of providing safety information. The paradigm also requires users attention, which could endanger them while driving.

## 1.4 Thesis Contribution

In this thesis, we address limitations of mobile devices by demonstrating that a combination of novel ubiquitous sensing and machine learning can extract useful information and build new applications to promote seamless safety and efficiency in mobility systems. As shown in Fig. 1.1, we focus on designing effective feedback loops for various mobility tasks. Our findings contribute to the mobility ecosystem from the perspectives of vehicles, traffic, and drivers.

### 1.4.1 From the Vehicle's Perspective: `V-Sense`

Detecting how a vehicle is steered and then alerting drivers in real time is paramount to vehicle and driver safety, as fatal accidents are often caused by dangerous steering. Solutions for detecting dangerous maneuvers tend to be available either exclusively in high-end vehicles or on smartphones as mobile applications. However, most solutions rely on the use of cameras, the performance of which is seriously constrained by high visibility requirements. Moreover, a sole or overreliance on cameras may distract drivers.

To address these problems, we have developed a vehicle-steering detection middleware called `V-Sense` that can run on commodity smartphones without additional sensors or infrastructure support. Rather than using cameras, the core of `V-Sense` "observes" a vehicles steering via non-vision smartphone sensors. We have designed and evaluated algorithms to detect and differentiate vehicle maneuvers such as lane changes, turns, and driving on winding roads. Because `V-Sense` does not rely on cameras, its vehicle-steering detection is not affected by the (in)visibility of road objects or other vehicles. We first detail the design, implementation, and evaluation of `V-Sense` and then demonstrate its practicality through two prevalent use cases: camera-free steering detection and fine-grained lane guidance. Our extensive evaluation results indicate that `V-Sense` can accurately determine and distinguish between various steering maneuvers, highlighting the middlewares utility for a range of safety-assistance applications without the need for additional sensors or infrastructure.

### 1.4.2 From Driver's Perspective 1: Dri-Fi

Our next project explores drivervehicle interactions and presents Dri-Fi, a solution that enables automotive apps to verify the person behind the wheel on the basis of mobile sensors. Driver identification is essential to personalized service/assistance for a driver and his/her designated parties. Many automotive apps can benefit from this capability: usage-based auto insurers; Apple CarPlay and Android Car, both of which connect smartphones with cars infotainment systems; and ridesharing services such as Uber and Lyft.

Dri-Fi contains several novel characteristics and achieves driver identification solely based on embedded IMUs on mobile devices. The core algorithm of Dri-Fi analyzes IMU data containing vehicles turning maneuver(s). It extracts three new features from raw IMU data and feeds them to train a machine learning model for driver identification. Various automotive apps can benefit from the information unearthed by Dri-Fi.

### 1.4.3 From Traffic's Perspective: TurnsMap

To explore mobile devices potential to understand traffic, we investigate the use of crowdsensing mobile data in transportation systems. Specifically, left turns are known to be the most dangerous driving maneuver at intersections due to oncoming traffic and pedestrian crossings. The danger of left turns can be reduced by implementing protected traffic phases exclusively for left turns, such as left-turn lights and 4-way stop signs. Although protected left turns are much safer, corresponding detailed information (e.g., whether an intersection has a protected left-turn phase) is not yet available to the public and/or automotive apps, such as navigation systems.

To extract such safety-critical information, we present TurnsMap, a framework for inferring left-turn information powered by two main components: (1) a mobile crowdsensing front end, which collects IMU sensor data (e.g., via gyroscope and accelerometer) from mobile devices carried by the driver/passengers in a moving car; and (2) the data analytics back end, used to infer left-turn information from IMU data collected via mobile crowdsensing. We also demonstrate that an array of apps in the automotive ecosystem can benefit from left-turn information provided by TurnsMap (e.g., to enhance traffic safety). We have evaluated this framework using large-scale real-world driving data, thereby demonstrating its ability to identify left-turn information with high accuracy at low cost.

### 1.4.4 From the Driver's Perspective 2: TurnsGuard

TurnsGuard explores mobile sensing to help drivers develop safer driving habits. In this case, we focus on raising peoples awareness of poor driving behavior, namely turning and/or changing lane(s) without using a turn signal, which is a leading cause of traffic accidents.

We have therefore developed TurnsGuard, an automatic unsignaled vehicle turn detection system that can be implemented on off-the-shelf devices. TurnsGuard can sense a drivers steering maneuvers and turn signal usage, which can then be used to determine the drivers driving pattern. Several sensing challenges must be addressed to detect unsignaled vehicle steering. To achieve ubiquitous turn detection, TurnsGuard uses motion sensors (i.e., a gyroscope and accelerometer) to detect turning and steering maneuvers. To detect turn signal usage without seeing the turn signal light, we will leverage microphones to identify periodic clicking (i.e., the tick-tock of the turn signal that is audible inside of the car). These design principles enable TurnsGuard to run locally on various sensing platforms. For example, TurnsGuard can operate on a drivers smartphone after the user installs an app. Design features also enable detection of an absent turn signal with minimal constraints; the driver does not have to fix the phones position, such as by using a dedicated phone mount.

## 1.5 Thesis Structure

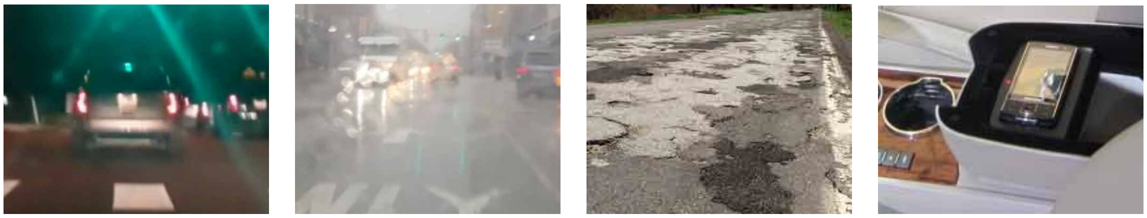
Chapter II of this thesis presents how ubiquitous sensors can be used to detect steering maneuvers regardless of disturbances to sensing devices (V-Sense). Chapter III demonstrates how, by focusing on turning maneuvers, we can characterize drivers driving patterns using a quantifiable metric (Dri-Fi). In Chapter IV, we show how microscopic analyses of crowdsourced ubiquitous sensory data can be used to infer critical macroscopic contextual information, such as risks present at road intersections (TurnsMap). Chapter V presents ubiquitous sensors to profile a drivers behavioral patterns on a large scale, which is crucial for analyzing and improving drivers driving behavior (TurnGuard). Finally, we discuss future research directions and conclude this thesis in Chapter VI.

# CHAPTER II

# V-Sense: Invisible Sensing of Vehicle Steering on Smartphones

Detecting how a vehicle is steered and then alarming drivers in real time is of utmost importance to the vehicle and the driver's safety, since fatal accidents are often caused by dangerous steering. Existing solutions for detecting dangerous maneuvers are implemented either in only high-end vehicles or on smartphones as mobile applications. However, most of them rely on the use of cameras, the performance of which is seriously constrained by their high visibility requirement. Moreover, such an over/sole-reliance on the use of cameras can be a distraction to the driver.

To alleviate these problems, we develop a vehicle steering detection middleware called `V-Sense` which can run on commodity smartphones without additional sensors or infrastructure support. Instead of using cameras, the core of `V-Sense` senses a vehicle's steering by only utilizing non-vision sensors on the smartphone. We design and evaluate algorithms for detecting and differentiating various vehicle maneuvers, including lane-changes, turns, and driving on curvy roads. Since `V-Sense` does not rely on use of cameras, its detection of vehicle steering is not affected by the (in)visibility of road objects or other vehicles. We first detail the design, implementation and evaluation of `V-Sense` and then demonstrate its practicality with two prevalent use cases: camera-free steering detection and fine-grained lane guidance. Our extensive evaluation results show that



(a) Lighting (b) Weather (c) Pavement (d) Placement

Figure 2.1: Visibility distortion under different conditions

V-Sense is accurate in determining and differentiating various steering maneuvers, and is thus useful for a wide range of safety-assistance applications without additional sensors or infrastructure.

## 2.1 Introduction

Automobiles bring a wide range of conveniences as well as fatalities. In 2012, the reported number of fatalities from road accidents was 30,800 in the U.S. alone [13]. Of these fatalities, 23.1% involved lane control — i.e., merging or changing lanes or driving on curvy roads — and 7.7% involved turning maneuvers, i.e., turning left/right or making U-turns. In total, 30.8% of the fatalities were related to vehicle steering maneuvers.[1]

As most of these fatalities had resulted from the driver's careless or erroneous steering, those accidents could have been minimized or prevented if effective safety mechanisms had been deployed in the vehicles. There has been an on-going push for incorporating electronic safety features in the vehicles to assist drivers' steering.

Steering-assistance systems, such as lane-departure warning or lane-keeping assistance, are the typical examples. They all exploit the advanced built-in sensors (e.g., cameras, radars, and infrared sensors) to detect the lane for driving assistance [46]. However, since they require special sensors which are only available on recent high-end cars, such safety solutions cannot be applied to a wide range of type/year models of cars.

To overcome such limitations, instead of using built-in vehicle sensors, efforts are being

[1] We refer to *steering maneuvers* as either changing lanes, turning left/right, or driving on curvy roads.

made to exploit the sensors in smartphones to assist drivers in their steering maneuvers. At one end of the spectrum of such applications, cameras have been used widely. The front/rear cameras of a smartphone are exploited to capture the images of road objects (e.g., traffic lanes, curb, and the preceding vehicle) which are then analyzed with image processing [28, 107, 178]. Although such systems claim that smartphone cameras are sufficient in assisting the driver, they have limitations in terms of computational overhead and inaccuracy. The accuracy of camera-based approaches depends on *visibility* and can thus be infeasible, depending on the conditions listed below and shown in Fig. 2.1.

- **Lighting**: The functionality of camera-based approaches cannot be guaranteed in case of insufficient light, especially at night time.
- **Weather**: Rainy or snowy weather will make roads waterly or icy, and will thus distort the light reflection, rendering it difficult to identify road objects.
- **Pavement**: Bad pavement conditions will distort the shape of road objects and will thus cause false or miss detections.
- **Camera placement**: Placing the phone at a location where the camera cannot capture the road objects (e.g., in the driver's pocket), will diminish the feasibility of the camera-based approach.

The other end of the spectrum is to not use cameras. Smartphone sensors, such as gyroscope, accelerometer, magnetometer, etc., can be exploited to detect the vehicle steering maneuvers and thus perform the same steering-assistance functionalities that would have been achieved with use of cameras [136, 170]. These approaches have advantages of requiring much less computational resources and power, and also being immune to visibility distortions. However, it is known to be very difficult to differentiate the steering maneuvers, which is one of the main reasons for camera-based approaches being most prevalent.

In this project, we propose `V-Sense`, a novel vehicle steering sensing middleware on smartphones, which overcomes the limitations and difficulties inherent in the existing camera-based and camera-free approaches. V-Sense is a camera-free middleware that can be utilized for various applications. It utilizes built-in Inertial Measurement Units (IMUs) on smartphones to detect various steering maneuvers of a vehicle. Specifically, V-Sense determines the changes in the angle of vehicle heading (i.e., steering) and the corresponding displacement during a steering maneuver. V-Sense classifies steering maneuvers into different types, such as turn, lane change, driving on curvy roads, etc., and exploits the classified results for various applications. We will elaborate on these applications in the following sections.

The contributions of this project are three-fold:

- Design of V-Sense, an all-time vehicle steering sensing middleware which does not rely on use of cameras;
- Detection and differentiation of various steering maneuvers by only utilizing a smartphone's built-in sensors; and
- Proposal of two driving-assistance applications — i.e., careless steering detection and fine-grained lane guidance — which are based on V-Sense functionalities.

The remainder of this chapter is organized as follows. Sec. 2.2 describes the motivation behind the design of V-Sense. Sec. 2.3 details the overall system and functionalities of V-Sense. We first evaluate the performance of V-Sense in Sec. 2.4 and then show two different applications of V-Sense in Sec. 2.5 and Sec. 2.6. After reviewing the related work in Sec. 2.7, we finally conclude this chapter in Sec. 2.8.

## 2.2 Motivation

Without relying on use of cameras and by only utilizing non-vision sensors on commodity smartphones, V-Sense can detect various steering maneuvers, such as left/right turns, changing lanes, or driving on curvy roads. How could such functionalities of V-Sense without using cameras at all, assist the driver in terms of both convenience and safety? Can it actually help in reducing fatalities from road accidents? Given below are two proof-of-concept applications of `V-Sense` that enhance safety and convenience of the driver's steering.

**Careless steering detection** Careless steering — changing lanes or making turns *without* using the turn signal on the car — is one of the main reasons for steering-related fatalities. A study from the Society of Automotive Engineers (SAE) unveiled that people in the U.S. forget to use their turn signals 2 billion times each day in total, or roughly 750 billion times per year. Based on such figures, SAE argued that about 2 millions of accidents can be prevented by eliminating turn signal neglects or careless steering [156]. In this application, `V-Sense` provides the functionalities required for detecting such careless maneuver. By combining lane-change detection via `V-Sense` and turn signal sound detection — which is designed based on a matched filter — the application determines whether the steering maneuver was accompanied with a turn signal, i.e., detecting whether the steering was careless or not.

**Fine-grained lane guidance** Existing navigation applications (e.g., Google map) provide information on *which* lane to stay on for preparing the next maneuver, i.e., indicating the correct lane. However, they lack functionalities of telling *whether* the vehicle is actually on that lane. If the driver fails to stay on the correct lane before its next maneuver, s/he would have to reroute or, in the worst case, take abrupt and thus dangerous lane changes to that lane. In order to provide assist the driver to determine whether the vehicle is on the correct lane, `V-Sense` provides the functionalities for fine-grained lane guidance. By integrating `V-Sense` and existing navigation systems, one can determine which lane the driver is currently on, and whether the lane is correct or not, without using cameras.

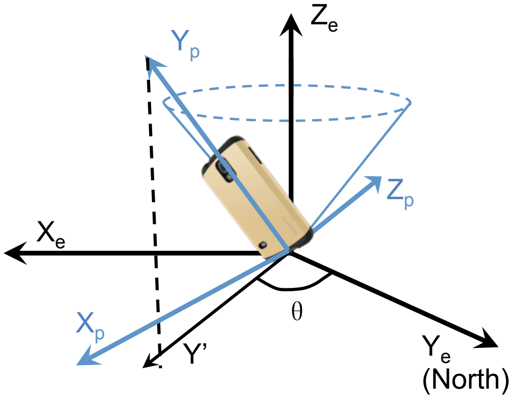


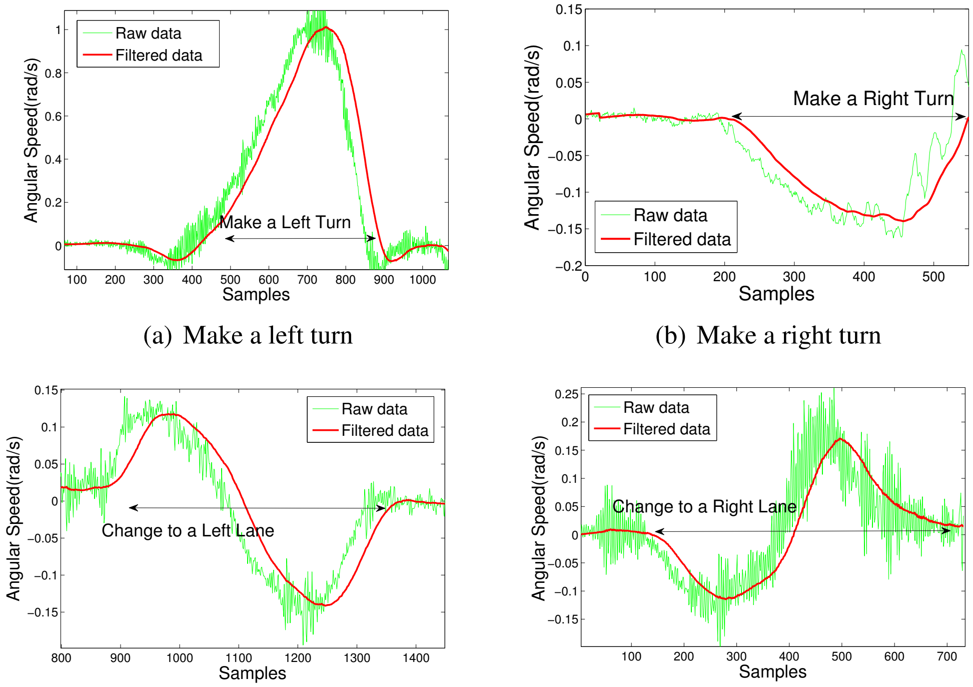
Figure 2.2: Align the phone coordinate system with the geo-frame coordinate system. This figure was borrowed from [181].

## 2.3 System Design

This section details the design and functionalities of `V-Sense`. First, we describe how IMUs on smartphones are utilized to determine whether the vehicle is making turn, changing lane, or driving on a curvy road. Then, we show how `V-Sense` classifies such different vehicle steering maneuvers based on the detection results.

### 2.3.1 Coordinate Alignment

Since the phone's coordinate changes over time, in order to maintain the consistency of analysis, we align the smartphone coordinate ($\{X_p, Y_p, Z_p\}$) with the geo-frame coordinate ($\{X_e, Y_e, Z_e\}$), as shown in Fig. 2.2. This allows us to simplify the change of the readings from 3 degrees of freedom (DoFs) to 1 DoF. The key idea is that with the measurements of the direction of the applied gravity to the smartphone ($Y$), the smartphone coordinate can be fixed within a cone. Then, combining the result with the angle ($\theta$) derived from the magnetometer readings and the thus-determined rotation matrix, the smartphone coordinate can be aligned with the geo-frame coordinate. We refer interested readers to [181] for detailed formulation of the rotation matrix.



(a) Make a left turn

(b) Make a right turn

(c) Change to a left lane

(d) Change to a right lane

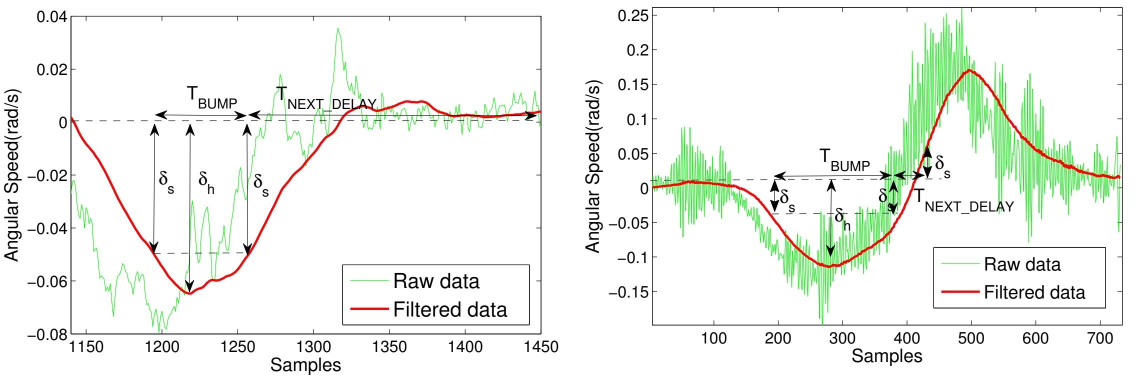
Figure 2.3: Gyroscope readings when the vehicle makes a left/right turn or left/right lane changing.

### 2.3.2 Bump Detection

When a car changes its direction via steering (e.g., changing lanes, making turns, and driving on curvy roads), the Z-axis gyroscope reading (i.e., yaw rate reading) on the phone can be utilized to represent the vehicle angular speed of that change of direction. Fig. 2.3 illustrates the Z-axis gyroscope measurements from the phone during a right/left turn, and changing to a right/left lane, respectively. During a left turn, a counter-clockwise rotation around the Z-axis occurs and thus generates positive readings (i.e., a positive bump), whereas during a right turn, a clockwise rotation occurs and thus generates negative readings (i.e., a negative bump).[2] Similarly, during a left lane change, a positive bump is followed by a negative bump, whereas during a right lane change, the opposite occurs.

Based on this observation, we can infer that by detecting *bumps* in the Z-axis gyroscope readings, we can determine whether the vehicle has made a turn or has changed a lane. The

[2] We refer to such a temporal rise/drop, or vice versa, of the yaw rate as *bumps*.



(a) One bump: a turn

(b) Two consecutive bumps: lane change

Figure 2.4: Statistical features of bumps in gyroscope reading during different steering maneuvers

other steering maneuver, i.e., driving on a curvy road, will show a similar *shape* but with a different *size* in terms of width and height of the bumps. We will elaborate on how `V-Sense` differentiates such steering maneuvers in Section 2.3.3.

We adopt a moving average filter to remove noise from the raw gyroscope readings. The delay parameter of the filter is set to 60 samples which correspond to 0.05 second in the time domain. Such a decision was made based on our experimental observation: it is short but good enough to extract the waveform of the bumps.

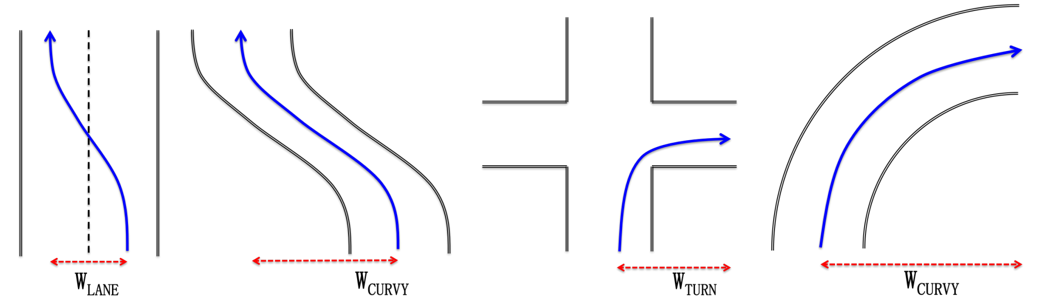
As shown in Fig. 2.4, we define four system parameters: $\delta_s, \delta_h$, $T_{BUMP}$, and $T_{NEXT\_DELAY}$. To reduce false positives and differentiate the bumps from jitters, a bump should satisfy the following three constraints for its validity: (1) all the readings during a bump should be larger than $\delta_s$, (2) the largest value of a bump should be no less than $\delta_h$, and (3) the duration of a bump should be no less than $T_{BUMP}$.

Based on these constraints of a valid bump, we designed an algorithm as shown in Algorithm 1, which keeps running when `V-Sense` operates. There are three states in the bump detection algorithm: *No-Bump*, *One-Bump*, and *Waiting-for-Bump*.

In *No-Bump* state, we continuously monitor the Z-axis gyroscope readings, i.e., yaw rate. When the absolute value of the measured yaw rate reaches $\delta_s$, we interpret this as the start of a possible bump and the algorithm enters *One-Bump* state.

**Algorithm 1** Algorithm for Detecting Bumps

```
Inputs:
    State, $Y$ (Yaw rate), System parameters
if State = No-Bump and —$Y$— ¿ $\delta_s$ then
    (Start of 1st bump)
    State ← One-Bump
    Record the start point of a possible bump
else if State = One-Bump and —$Y$— ¡ $\delta_s$ then
    Record the end point of a possible bump
    if bump is valid then
        State ← Waiting-for-Bump
    else
        State ← No-Bump
    end if
else if State = Waiting-for-Bump then
    $T_{dwell}$ ← State dwell duration
    if $T_{dwell} < T_{NEXT\_DELAY}$ and —$Y$— ¿ $\delta_s$ then
        (Start of 2nd bump)
        if 2nd bump is valid then
            Two valid bumps → “Lane change”
        else
            One valid bump → “Turn”
        end if
        State ← No-Bump
    else if $T_{dwell} > T_{NEXT\_DELAY}$ then
        One valid bump → Turn
        State ← No-Bump
    else
        Continue in Waiting-for-Bump state
    end if
else
    Continue in current state
end if
```



(a) Lane change (b) A S-shaped curvy road (c) Turning (d) A L-shaped curvy road

Figure 2.5: The same vehicle trajectory shape for four different scenarios: (a) lane change, (b) driving on an S-shaped curvy road, (c) turning, and (d) driving on an L-shaped curvy road.

The *One-Bump* state terminates when the yaw rate drops back to a value below $\delta_s$. If the sojourn/dwell time in *One-Bump* state was larger than $T_{BUMP}$ and the largest measured yaw rate was larger than $\delta_h$, hence satisfying the three constraints, we consider the first detected bump to be valid. In such a case, the algorithm enters *Waiting-for-Bump* state, Otherwise, it returns to *No-Bump*.

In *Waiting-for-Bump* state, it further monitors the yaw rate readings for a maximum dwell time $T_{NEXT\_DELAY}$. In the meanwhile, if another bump starts, i.e., the yaw rate reaching $\delta_s$ with a sign opposite to the first bump's is detected, it goes through the same procedure as before in validating it. If determined as valid, this would mean that two consecutive bumps with opposite signs have been detected. Thus, the algorithm determines the maneuver to be a *lane change*. Otherwise, if the second bump turns out to be invalid, then it would mean that only a single valid bump was detected and thus the algorithm determines the maneuver to be a *turn*. After making all decisions, the algorithm goes back to the initial *No-Bump* state.

The bump-detection algorithm is executed iteratively for each collected sample, and goes through a different procedure depending on the current state.

### 2.3.3 Differentiating Steering Maneuvers

When the vehicle is steered, bumps in the yaw rate readings are constructed. Based on the bump detection algorithm, `V-Sense` detects such bumps and differentiates between maneuvers of a lane change and a turn.

One possible problem in using this would be when driving on a curvy road. As illustrated in Fig. 2.5, when driving on a curvy road, it might have the same *shape* of trajectory as in lane change or a turn, and hence construct the same number and shape of bumps. In such a case, `V-Sense` might misinterpret the drive on a curvy road as a lane change or turn, thus yielding false positives/negatives. Therefore, it is imperative for `V-Sense` to differentiate between lane changes, turns, and also driving on curvy roads.

We achieve this by classifying the maneuvers based on not only the number and shape of the bumps as in Algorithm 1, but also with their horizontal displacement.[3] Let $W_{LANE}$ denote the horizontal displacement after a lane change. Since the average lane width is around 3.65 meters [166], $W_{LANE}$ is expected to be around that value after a lane change. In contrast, while driving on a curvy road, the horizontal displacement, denoted as $W_{CURVY}$, is usually much larger than $W_{LANE}$. Based on this observation, if `V-Sense` has detected two bumps — which means a possible lane change — it then derives the horizontal displacement during that steering maneuver. If the derived value is larger than 3.65 meters, `V-Sense` determines the vehicle to be driving on a curvy road, rather than making a lane change.

Also, to differentiate between turns at the intersection (i.e., a sharp turn) and driving on a curvy road, we exploit the fact that the horizontal displacement during a turn is much smaller than that during driving on a curvy road. Figs. 2.5(c) and 2.5(d) illustrate this, where $W_{TURN}$ and $W_{CURVY}$ represent the horizontal displacements during a turn and driving on a curvy road, respectively. If `V-Sense` detects only one bump, it further examines the

[3] *Horizontal displacement* is a value that represents the change of position in the X-axis after the steering maneuver.

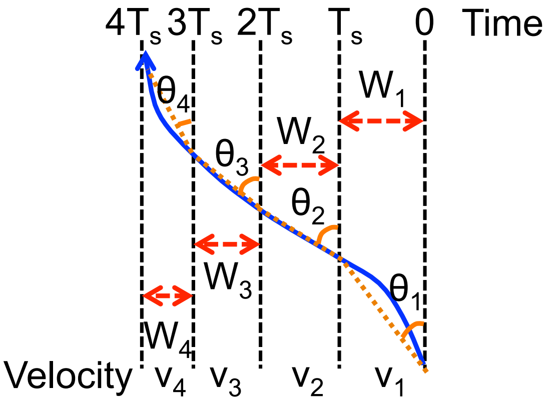


Figure 2.6: Deriving the horizontal displacement based on gyroscope readings and estimated velocity

horizontal displacement to distinguish between turning and driving on a curvy road.

Note that to differentiate turns from lane changes, only the number and shape of the bumps are required, which can be met by the bump-detection algorithm.

### 2.3.4 Horizontal Displacement

In order to correctly distinguish between lane change, turn, and driving on a curvy road, we must determine the horizontal displacement, in addition to the detection of bumps. We derive the horizontal displacement from the readings of a smartphone's gyroscope and accelerometer.

Fig. 2.6 shows an example vehicle trajectory during a left lane change or maneuver on a curvy road as illustrated in Figs. 2.5(a) and 2.5(b). The dotted vertical line represents the time when the sensors are sampled with frequency of $1/T_s$. Here $\theta_n$ denotes the angle of the vehicle's heading, whereas $v_n$ represents the average velocity during the sampling period. During each sampling period $T_s$, the vehicle's horizontal displacement can be expressed as:

$$W_n = v_n T_s sin(\theta_n). \tag{2.1}$$

Since the yaw-rate readings from the gyroscope represent the vehicle's angular velocity around the Z-axis, $\theta_n$ can be expressed as:

$$\begin{aligned}\theta_n &= \theta_{n-1} + Y_{avg}T_s \\ &\approx \theta_{n-1} + Y_nT_s,\end{aligned} \tag{2.2}$$

where $Y_{avg}$ represents the average yaw rate during the sampling period, and $Y_n$ the instantaneous yaw rate measured at the end of the sampling period. Note that the above approximation holds since the sampling period on smartphones can be significantly reduced. Thus, the total horizontal displacement from time 0 to $NT_s$ can be derived as:

$$\begin{aligned}W_{final} &= \sum_{n=1}^{N} W_n \\ &= \sum_{n=1}^{N} v_nT_s sin(\theta_n) \\ &= \sum_{n=1}^{N} v_nT_s sin(\sum_{k=1}^{n} Y_kT_s)\end{aligned} \tag{2.3}$$

where $T_s$ is a predefined parameter denoting the sampling period of the application. The third equality comes from the fact that the initial angle of the vehicle's heading, $\theta_0 = 0$, since this is the reference point. $Y_k$ can be acquired from the gyroscope readings, while $v_n$ can be derived from the accelerometer and GPS readings. We further elaborate on how to obtain an accurate value of $v_n$ in Section 2.3.6.

We exploit the gyroscope and accelerometer readings to determine $W_{final}$. Then, by analyzing the thus-determined value, `V-Sense` distinguishes between the cases of lane change or turn and driving on curvy roads. Using in-depth evaluations, we will later show that the derivation of horizontal displacement is accurate for various cases (e.g., lane change, left/right turn, U-turn).

### 2.3.5 Change in Vehicle's Heading Angle

Based on bump detection and horizontal displacement, `V-Sense` classifies various steering maneuvers into three classes: lane change, turn, and driving on a curvy road. To further classify different turning maneuvers (e.g., left/right turn at the intersections, U-turn), `V-Sense` derives the change in the vehicle's heading angle, i.e., the difference in the heading angle between the start and the end of a steering maneuver.

As in Eq. (2.2), the angle of vehicle's heading at sampling time $nT_s$ can be derived by accumulating the $n$ yaw-rate measurements. As an example, consider Fig. 2.6; at sampling time $3T_s$, the angle of the vehicle's heading would be $\theta_3 = \sum_{n=1}^{3} Y_n T_s$. In other words, the change in the vehicle's heading from time 0 to $NT_s$ can be expressed as:

$$\theta_{final} = \sum_{n=1}^{N} Y_n T_s. \tag{2.4}$$

For example, after making a left/right turn at the intersection, $\theta_{final} \approx \pm 90^\circ$, whereas after making a U-turn, $\theta_{final} \approx \pm 180^\circ$. Thus, by exploiting the derived values, `V-Sense` can further classify the turns into a left/right turn or a U-turn.

Fig. 2.7 summarizes the overall maneuver classification in `V-Sense` as a state diagram. `V-Sense` first determines whether the steering maneuver is a turn or a lane change by calculating the number of bumps. If it is a turn, `V-Sense` will calculate the angle change to determine whether the turn is a regular left/right turn or a sharp U-turn. Second, `V-Sense` calculates the horizontal displacement to determine whether it is an curvy road or not.

### 2.3.6 Velocity Estimation

In order to derive the horizontal displacement and set $T_{BUMP}$ and $T_{NEXT\_DELAY}$, we need accurate measurement of the vehicle's instantaneous velocity.

There are two ways of acquiring the velocity with a smartphone: reading the Speed Over Ground (SOG) output from the GPS module inside the smartphone, or exploiting the

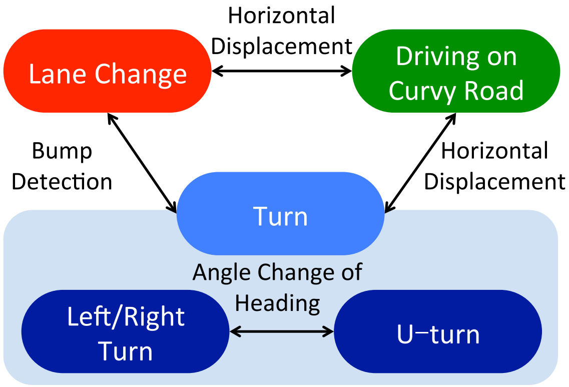


Figure 2.7: State diagram of maneuver classification in `V-Sense`.

IMU. The GPS does provide measurements of the velocity, whereas the acceleration can be derived from IMU readings. However, the GPS output rate is very low, e.g., 1Hz on Samsung Galaxy S4, as shown in Fig. 2.8, and hence cannot properly capture velocity changes within a sampling period. On the other hand, the IMU has a much higher output rate but contains lots of noise as well as some biases as shown in Fig. 2.8. Thus, just simply using either the velocity measurement from GPS or taking an integral of the accelerator IMU output is not sufficient. Hence, in order to exploit the distinct advantages of GPS and IMU, we fuse the data by using a Kalman filter [101] to estimate the velocity.

We first construct a model for estimating the velocity:

$$v(k|k-1) = v(k-1|k-1) + (a(k) - b(k-1|k-1))T_s \tag{2.5}$$

where $v(k|k-1)$ is the estimated velocity at time $k$ based on the optimized velocity at time $k-1$; $v(k-1|k-1)$ is the optimized velocity at time $k-1$; $a(k)$ is the acceleration output at time $k$; $b(k-1|k-1)$ is the optimized bias of the accelerometer at time $k-1$; $T_s$ is the sampling period of the accelerometer.

Here we treat $b$ as a constant bias [116]:

$$b(k|k-1) = b(k-1|k-1). \tag{2.6}$$

Thus, we have a matrix representation of the model as:

$$X(k|k-1) = AX(k-1|k-1) + BU(k) \tag{2.7}$$

where $X = \begin{bmatrix} v \\ b \end{bmatrix}$, $A = \begin{bmatrix} 1 & -T_s \\ 0 & 1 \end{bmatrix}$, $B = \begin{bmatrix} T_s \\ 0 \end{bmatrix}$, and $U$ is the output from accelerometer. So, the covariance matrix is estimated by:

$$P(k|k-1) = APA^T + Q, \quad Q = \begin{bmatrix} q_v & 0 \\ 0 & q_a \end{bmatrix} \tag{2.8}$$

where $P$ is the covariance matrix, and $Q$ is the covariance of the process noise which can be regarded as the Gaussian white noise. Thus, the state can be estimated as:

$$X(k|k) = X(k|k-1) + g(k)(S(k) - HX(k|k-1)) \tag{2.9}$$

where $g(k)$ is the matrix of Kalman gain and $S(k)$ is the speed relative to the ground measured by the GPS, and $H = [1 \quad 0]$. We refer the interested readers to [101] for more details.

Fig. 2.8 shows velocity estimation by using such a model based on Kalman Filter. Here we get the ground truth velocity by directly reading it from the OBD-II port, and compare it with our estimation results. Fig. 2.8 shows that the velocity can be accurately estimated in real time, thus yielding accurate horizontal displacements.

### 2.3.7 Parameter Setting

The bump-detection algorithm uses four main parameters: $\delta_s$, $\delta_h$, $T_{BUMP}$ and $T_{NEXT\_DELAY}$. $\delta_s$ determines the start/end point of a bump and thus is the smallest reading value, whereas $\delta_h$ determines the largest reading value of the bump, and hence represents its height. Based on the constraints of a *valid* bump, its minimum and maximum should be

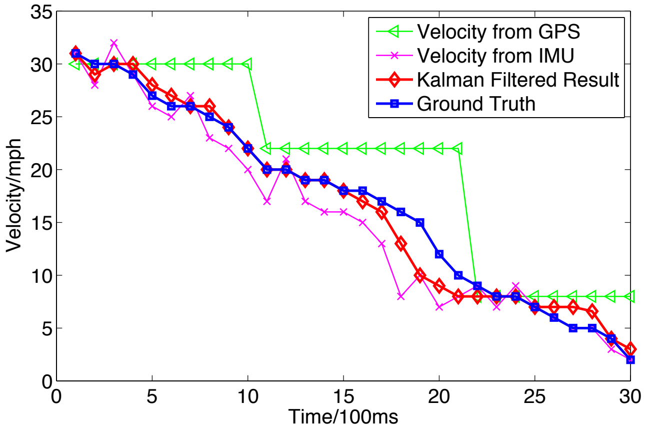


Figure 2.8: The accuracy of velocity estimation by fusing sensor readings

larger than $\delta_s$ and $\delta_h$, respectively.

With large values of $\delta_h$ and $\delta_s$, small bumps — which may be caused by background noise or sensing errors — can be ignored and thus reduce the false-positive rate, whereas the false-negative rate might increase. On the other hand, with small values of $\delta_h$ and $\delta_s$, the false-negative rate can be reduced but will become susceptible to background noise, thus increasing the false-positive rate. From extensive road tests, we found that parameters of $\delta_s = 0.05$ and $\delta_h = 0.07$ represent a good tradeoff, and are thus used as default values for simplicity. However, the optimal parameter setting may slightly vary with the driving habit. Developing an adaptive parameter selection mechanism is part of our future work.

As for the other two parameters, $T_{BUMP}$ represents the time duration of a valid bump, whereas $T_{NEXT\_DELAY}$ represents the maximum waiting time for the following bump, in case of a lane change. Since the time duration of a turn or lane change is usually several seconds [130], we set $T_{BUMP} = 1.5$ seconds and $T_{NEXT\_DELAY} = 3$ seconds as their default values.

| | **Lane Change** | | | | **U-turn** | | | |
|---|---|---|---|---|---|---|---|---|
| | **#1** | **#2** | **#3** | **Average** | **#1** | **#2** | **#3** | **Average** |
| **Displacement [m]** | 4.29 | 3.49 | 3.59 | 3.79 | 14.47 | 15.66 | 14.46 | 14.86 |
| **Angle Change [deg]** | 2.03 | 7.49 | 4.12 | 4.54 | 193.73 | 179.85 | 184.41 | 185.99 |

Table 2.1: Determined horizontal displacement and angle change of heading for lane changes/U-turns.

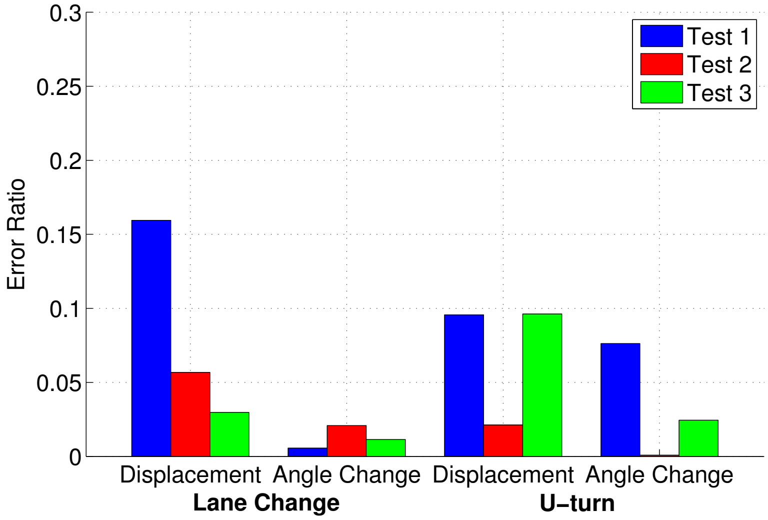


Figure 2.9: Error of the determined values compared with the ground truth.

## 2.4 Evaluation

To evaluate the performance of `V-Sense`, we implemented `V-Sense` on a Samsung Galaxy S4 with a 1.6GHz quad-core processor running Android 4.4.1 KitKat OS. We have conducted a total of 40 hours of test, and tried to cover different environments both in a parking lot and real roads. First, we evaluated the accuracy of `V-Sense` in determining the change of heading angle and the horizontal displacement in a short road test. Then, we evaluate the performance of `V-Sense`'s classification with a longer road test containing various road features. The cars we used for the test were a 2010 Mitsubishi Lancer and a 2006 Mazda 6. During these experiments, the smartphones were either mounted on the windshield, or kept in the driver's pocket.

### 2.4.1 Accuracy of Estimating Angle Change and Displacement

By making several lane changes, turns, and U-turns during a short road test, we evaluated the accuracy of `V-Sense` in estimating the change of heading angle and the horizontal displacement. During the road test, we made three lane changes, one to the left lane and the other two to the right lane, and three U-turns. We collected the horizontal displacements and changes of heading angle from V-Sense to check whether the estimated values are close to their ground truth. The results of the two separate tests are summarized in Table 2.1. For consistency, we present all numbers as their absolute values.

During a lane change, the ground truth horizontal displacement is expected to be equal to the actual lane width, which was around 3.7m for our experiment. However, for the change of heading angle, it is expected to be 0°, since this is a measure of the difference between the initial and the final heading angles.

On the other hand, during a U-turn, the ground truth of horizontal displacement and the change of heading angle are the road width for U-turns, which was approximately 16m in our case, and 180°, respectively.

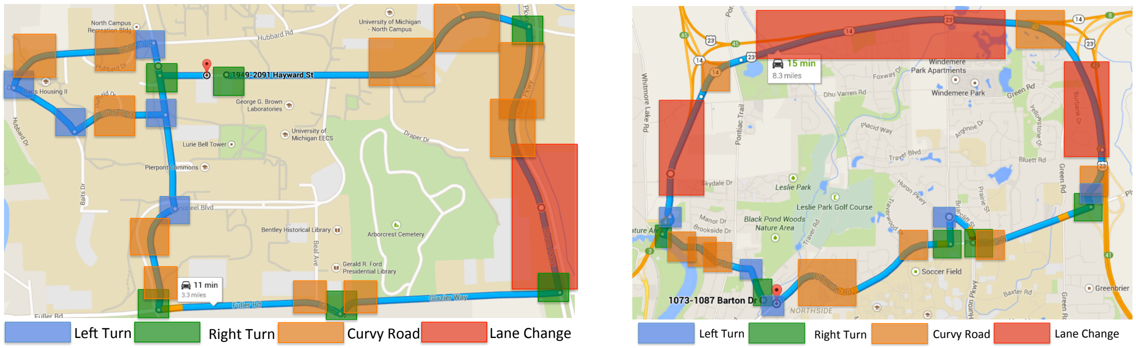
Fig. 2.9 shows the error ratio — which is the ratio of the absolute deviation to the ground truth value — in the two experiments. For all cases, the estimated horizontal displacement and change of heading angle have a very low error ratio, i.e., V-Sense is very accurate.

The high accuracy of V-Sense in determining the two values means that it can correctly classify various steering maneuvers, which is validated in the following subsection by conducting long road tests.

### 2.4.2 Accuracy of Maneuver Classification

To evaluate how well V-Sense classifies different steering maneuvers, we performed two long road tests. To guarantee the generality of our experiment, we carefully chose two different test routes as shown in Figs. 2.10 (a) and (b) near our campus. The routes run through typical urban areas and freeways, the former including residential, downtown, and school areas. The road features are highlighted in both figures with detailed information as follows.

- Left/right turn (LT/RT): Each turn at an intersection exemplifies a left/right turn.
- Curvy road (CR): There are several curvy roads in the chosen routes. Among them, there are two long L-shaped curvy roads on the US#23 freeway, which are



(a) Testing route #1

(b) Testing route #2

Figure 2.10: Real road testing routes used for evaluation in Ann Arbor, MI. Here testing route #1 is around campus, milage is 3.3 miles; testing route #2 is freeway, milage is 8.3 miles.

challenging for our bump detection scheme alone to determine.

- Multiple traffic lanes (LC): A lane change is possible in both urban areas and on the US#23 freeway. Specifically, there are 2 lanes in each direction in the urban road, and 4 lanes in each direction on the US#23 freeway.

The number of features in the examined routes is summarized in Table 2.2.

| Route | Distance [miles] | RT | LT | LC | CR |
|---|---|---|---|---|---|
| **#1** | 3.4 | 6 | 5 | 4 | 11 |
| **#2** | 8.3 | 5 | 5 | 15 | 9 |

Table 2.2: Summary of different road features in testing routes.

To validate the independence of V-Sense from driving habits, we had 5 volunteers participating in our test, three male drivers and two female drivers. Each of them drove twice on both route #1 and #2. In the first test, they mounted the phone on the windshield, whereas in the second test, the phone was kept inside the driver's pocket.

The on-road experimental results are plotted in Fig. 2.11, and can be highlighted as follows.

- V-Sense achieves 100% accuracy in detecting both right and left turns, regardless of the phone's placement and road condition. This is because when turning, the heights of the bumps in the readings tend to be high enough to be accurately detected and classified.

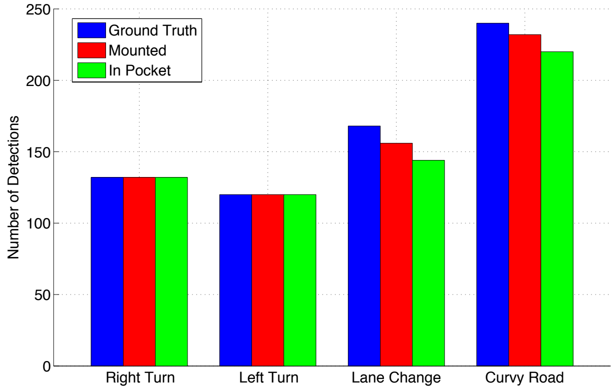


Figure 2.11: Performance of recognizing different steering patterns on both route #1 and #2

- For lane changes, `V-Sense` achieves 93% accuracy when the phone is mounted on the windshield, and 85% accuracy when the phone is in the driver's pocket. The false-negative results are mostly due to the fact that `V-Sense` occasionally misinterprets a lane change as driving on a curvy road, because a few of the lane changes in our test took longer than expected, especially on the freeway where drivers tend to take extra caution, thus making slower lane changes. The accumulated error in the gyroscope reading can also degrade the performance in such a case [181]. However, its occurrence is expected to be rare considering the average elapsed time for lane changing, i.e., less than 6 seconds.
- `V-Sense` achieves nearly 97% accuracy in detecting curvy roads with the phone mounted on the windshield, and nearly 92% accuracy with the phone kept in the driver's pocket. These results also reflect the accuracy of the coordinate alignment mentioned in Section 3.1. Also, note that `V-Sense` was able to detect the two long L-shaped curvy roads on the US#23 freeway using bump detection and horizontal displacement derivation.

### 2.4.3 `V-Sense` vs. Camera-Based Approach

We also compare the performance of `V-Sense`, a camera-free approach, with existing camera-based approaches. Since most of the existing driving assistant applications can

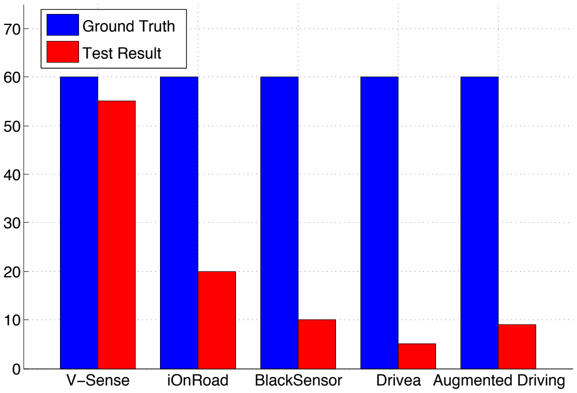
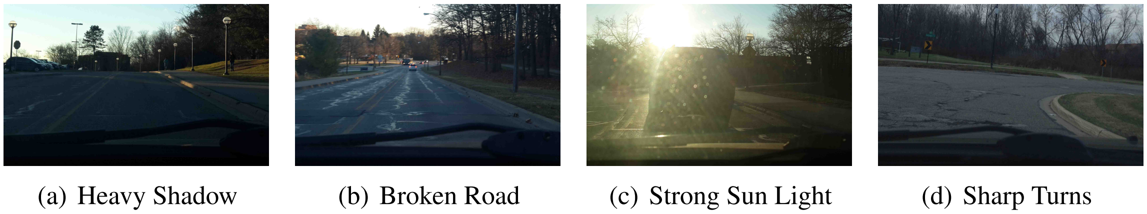


Figure 2.12: Comparison of V-Sense, iOnRoad, BlackSensor, Drivea, and Augmented Driving in lane-change detection.

only detect lane changes, not turns or driving on curvy roads, we do this comparison with only the results of lane-change detection. The evaluation is based on tests of both route #1 and route #2. We choose iOnRoad [28], BlackSensor [6], Drivea [12], and Augmented Driving [5] for comparison with V-Sense. All of these four apps have the capability of detecting lane departures. Of these apps, iOnRoad is the most popular one with more than 1,000,000 downloads from Google Play, and it is still under active maintenance, whereas Drivea is the least popular, still with more than 10,000 downloads.

Since the use of cameras is almost 100% ineffective at night or under a bad weather, we compared the performance of V-Sense with difference apps based on the camera approach, during daytime and under a perfect weather condition. Here we used two Samsung Galaxy S4 smartphones: phone *A* runs V-Sense while phone *B* is running either iOnRoad, BlackSensor, Drivea, or Augemented Driving. In all experiments, phone *A* was placed next to the driver's seat, while phone *B* was mounted on the car's windshield to have a clear view of the road.

As shown in Fig. 2.12, V-Sense achieves about 3× better accuracy than iOnRoad, 5× better than BlackSensor and Augmented Driving, and 11× better than Drivea. According to our experiments conducted even under a perfect weather condition, the performances of all the compared apps were still seriously constrained by the environment, while



(a) Heavy Shadow (b) Broken Road (c) Strong Sun Light (d) Sharp Turns

Figure 2.13: Non-functional environments for the camera-based driving assistant application in the experiment.

V-Sense worked well, irrespective of the environment. In particular, the performance of camera-based lane detection degraded severely in at least four cases: heavy shadow on the road, broken road, strong sunlight, and sharp turns as shown in Fig. 2.13.

### 2.4.4 Computational Cost of V-Sense

The high computational requirement is always the problem for existing driver assistant applications. For example, CarSafe [178] indicates its CPU utilization to be 64% when run on Samsung Galaxy S3. We evaluated the computational cost of V-Sense on smartphones, including Samsung Galaxy S3 (with 1.4GHz quad-core Cortex-A9 CPU), and Samsung Galaxy S4 (with 1.6GHz quad-core Cortex-A15 CPU). The CPU utilization was monitored by using adb top provided by Android SDK.

For a fair comparison with the camera-based approach, we extracted the lane change detection functionality and implemented a simple application only with the lane-change detection function. The techniques used in this application exploit the phone's rear camera to acquire the road's image, and implement a popular lane-detection algorithm [95] to extract the lane.

Our experimental results are plotted in Fig. 2.14. We found that, on Galaxy S4, the average CPU utilization of V-Sense was 16.9%, whereas for the lane detector, it was 28.4%. On Galaxy S3, the average CPU utilization of V-Sense was 38.1%, whereas for the lane detector, it was 60.6%. These results show that V-Sense uses 50% less CPU than the camera-based approach. In other words, V-Sense not only achieves higher accuracy

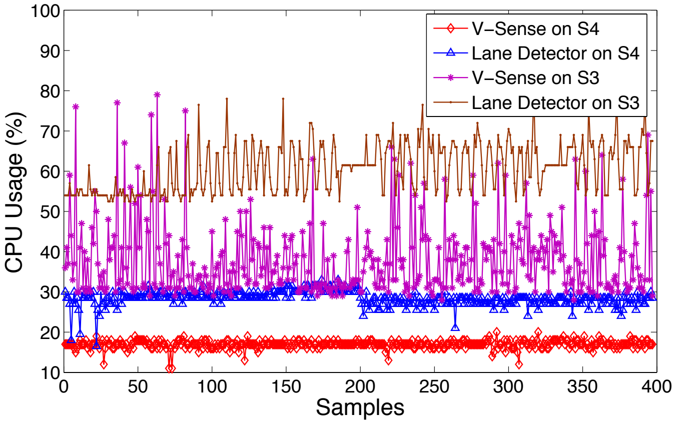


Figure 2.14: Comparison of CPU usage between V-Sense and lane detector

but also lower computational overhead than camera-based approaches.

## 2.5 Application I: Detection of Careless Steering

V-Sense can be used to detect careless steering: changing lanes or making turns without turning on the turn signal. Detecting a driver's careless steering is important, since it would enhance the safety of not only the driver but also people/vehicles around him. Moreover, it can also be used by insurance companies in monitoring the driver's driving habit and thus determining the insurance premium accordingly, which would then motivate the drivers to avoid careless steering.

In this section, we present and evaluate a simple proof-of-concept of careless steering detection. We use V-Sense to detect and differentiate various steering maneuvers, such as making turns, changing lanes, and driving on curvy roads. Furthermore, we present a scheme to detect whether the driver has actually used the turn signal during each steering maneuver.

### 2.5.1 Overview

Fig. 2.15 shows the information flow of careless steering detection using data collected from the gyroscope, GPS, accelerometer, and microphone on a smartphone. This

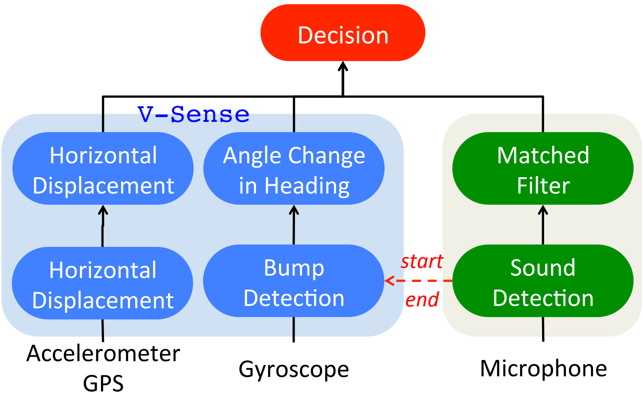


Figure 2.15: Information flow of careless steering detection by combining V-Sense and sound detection module.

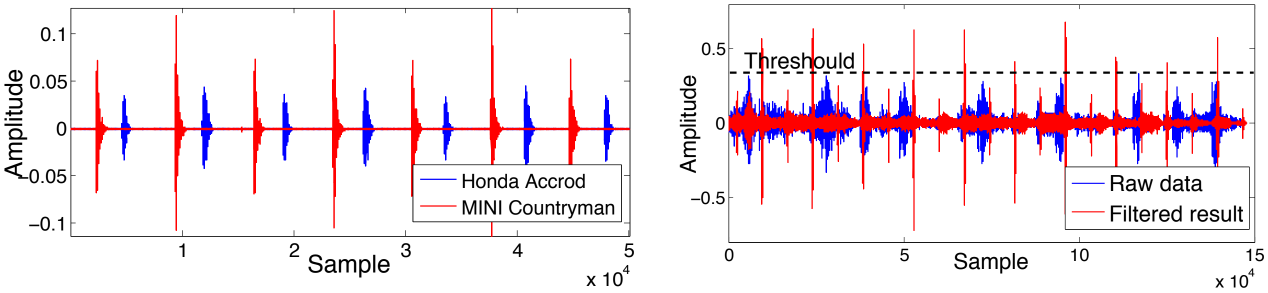
application is comprised of V-Sense and sound detection module. V-Sense detects possible lane changes or turns using the gyroscope readings. Upon detecting the start point of a lane change or turn, i.e. bump, the sound detection module activates the microphone and starts to detect the turn signal sound. If a lane change or turn is detected without the detection of signal sound, the application declares the driver is involved in careless driving, and triggers alarm to notify the driver. Otherwise, the application declares it as attentive driving.

### 2.5.2 Detection of Turn Signal

In order to detect whether the driver has used the turn signal, V-Sense uses the following three steps: (i) collect training samples of the turn signal sound; (ii) eliminate background noise with a matched filter; (iii) make a decision on whether the turn signal was used during the turn or lane change.

#### 2.5.2.1 Collection of Training Sample Data Set

We first collected the turn signal sounds from two different cars, 2006 Honda Accord and MINI Countryman, which are used as sample data sets as shown in Fig. 2.16(a). The measured steady rates of the turn signal in the 2006 Honda Accord and MINI Countryman



(a) Signal sample

(b) Matched filter result

Figure 2.16: Turn signal samples and the filtered result by using a matched filter, (a) turn signal sample from a 2006 HONDA Accord and MINI Countryman; (b) matched result from the filter with the existence of background noises inside the car.

were 161 and 163 ticks per second (shown in Fig. 2.16), respectively.

As the turn signal sounds acquired from the 2006 Honda Accord has lower amplitude, and would thus be more difficult to detect, we studied the sound detection module using this data set. To test the performance of our turn signal detection module in real driving scenario, we turned on the engine and played music which acts the background noise inside the car.

#### 2.5.2.2 Elimination of Noise with a Matched Filter

To detect the sound emitted from the turn signals, the detection module has to overcome two challenges: (i) it must be resilient to the variation of SNR due to unpredictable detection conditions; (ii) the delay in detecting a single turn signal must be low in order to be ready for detecting the subsequent signal. We utilized a matched filter to meet these challenges.

The matched filter is used to detect the presence of a known signal in the unknown signals [125]. The key idea behind the matched filter is to design an impulse response that maximizes the output SNR. Due to unpredictable driving conditions, the noise inside the car cannot be easily modeled. Thus, we model the turn signal and extract the signal sound by using one convolution with the matched filter kernel. Since the turn (sound) signal can be modeled as series of discrete signals, we use the discrete version of matched filter, in which the output, $y[n]$, can be expressed as:

$$y[n] = \sum_{k=-\infty}^{\infty} h[n-k]h[k], \tag{2.10}$$

where the impulse response, $h[n]$, of the matched filter is

$$h[n] = g[n] \otimes v[n_0 - n], \tag{2.11}$$

where $g[n]$ denotes the power spectral density of background noise. We can thus acquire the matched signal $Matched$ by applying

$$Result[n] = Signal[n] \otimes h[n], \tag{2.12}$$

where $signal[n]$ is the sound recorded by the smartphone's microphone inside the car and $Result[n]$ is the output of the matched filter.

#### 2.5.2.3 Making a Decision

If the amplitude of the matched filter output is larger than $T$, a pre-defined threshold set to 0.35 by default, `V-Sense` declares the detection of a turn signal sound. If the detected turn signal is accompanied by a lane change or turn detected by `V-Sense`, then the application declares the steering maneuver as being attentive. On the other hand, if no such turn signal sound was detected, the application declares the steering to be careless, and then alarms the driver.

### 2.5.3 Performance of Sound Detection

Fig. 2.16(b) shows the performance of our sound detection module which extract signal sound from background noise. We conducted experiments in a regular driving setting, where music played inside the car and the passengers were talking occasionally. The matched filter was able to extract and identify the sound of the turn signals from the background noise, even when the amplitude of the noise was very high (radio played music at the max volume).

By integrating this accurate sound detection module with V-Sense, the application detects careless steering and thus enhances driving safety significantly.

## 2.6 Application II: Fine-Grained Lane Guidance

In this section, we demonstrate a proof-of-concept fine-grained lane guidance application using V-Sense. Fine-grained lane guidance allows existing navigation systems provide higher guidance accuracy. Specifically, fine-grained lane guidance detects whether the driver is in the correct lane and alarms the driver if not.

Existing navigation systems on smartphones are constrained by the accuracy of the built-in GPS sensor, which is at best 5∼10m [148]. When the line-of-sight transmission between the satellite and the phone is blocked by obstacles, such as tunnels, bridges, and tall buildings, the accuracy quickly drops to 20∼100m [98]. Such limitations make it impossible for legacy navigation systems to recognize the exact traffic lane that the vehicle is on. The latest update of Google Maps does include a lane guidance function [17], but in a rather limited way: it can only provide information on which lane the vehicle should stay, not whether it is actually on that lane.

We incorporate V-Sense in an existing navigation system to provide a true fine-grained lane guidance. Fine-grained lane guidance is important, since it can reduce abrupt lane changes, and also very helpful for drivers who have lack driving experience.

### 2.6.1 Achieving Fine-Grained Lane Guidance

Based on information from an on-line map, the correct lane for the next maneuver can be easily determined, which is a function already provided by existing navigation systems.

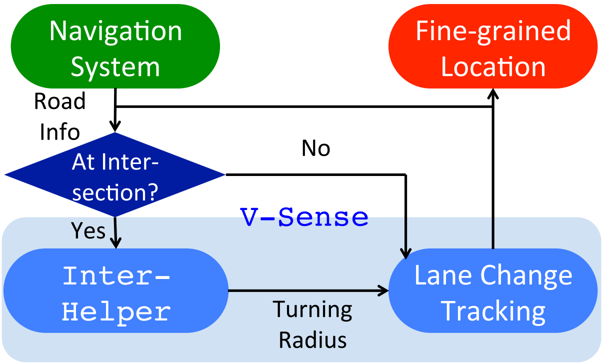


Figure 2.17: Information flow of fine-grained lane guidance by incorporating navigation system and V-Sense.

Hence, the main challenge of realizing the fine-grained lane guidance application is the determination of the current lane that the vehicle is running on. To meet this challenge, we need to determine the vehicle's current lane via lane change detection. The current lane can be determined based on whether and how the vehicle has changed its lane. Thus, by detecting and analyzing the lane changes made by the vehicle, we can determine the vehicle's current lane.

Lane changes may take place in two situations: (i) middle of a road or; (ii) at intersections.[4] For the first case, V-Sense can reliably detect lane changes on the road using techniques in Section 2.3. To implement accurate lane tracking for the second case, we develop an add-on module of V-Sense called InterHelper. In Fig. 2.17, we show how the navigation system and V-Sense cooperate to determine the fine-grained location. The navigation system is capable for determining whether the vehicle is at the intersection. Once the vehicle reaches an intersection, InterHelper is triggered and starts to estimate the turning radius, $R_i$. Note that $R_i$ is equivalent to the horizontal displacement during the turn, which can be derived by using the techniques described in Section 2.3.4. This information enable V-Sense to finally determine the fine-grained location.

As shown in Fig. 2.18, there are four possibilities of lane change at a typical 4-lane single carriageway intersection. That is, each car has two choices of making either right or

[4] Although it is illegal to change lane at intersections in some U.S. states (e.g., California [100]), we still consider such a case for generality.

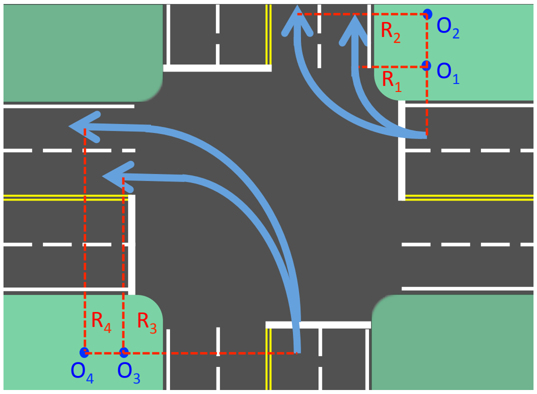


Figure 2.18: Turns and the corresponding turning radius at a 4-lane single carriageway intersection.

left turn. Here, we assume the turning trajectory is an arc, which is a common assumption in intersection design [25]. $O_1$, $O_2$, $O_3$ and $O_4$ are centers of turning circles. InterHelper classifies each case by differentiating the turning radius, i.e., $R_1$, $R_2$, $R_3$ and $R_4$.

For a typical intersection, the right turn radius, $R_1$ is 10.8m [168], the left turn radius, $R_3$ is 20.7m, and the width of a typical intersection is 19.2m [45]. Moreover, the lane width is around 3.65m [130]. Based on these facts and extensive road experiments, we set the threshold of differentiating $R_1$ and $R_2$ as 13.1m, and the threshold of differentiating $R_3$ and $R_4$ as 21.64m. Using such thresholds and the horizontal displacement obtained from V-Sense, the application determines whether the vehicle has changed its lane during a turn at the intersection.

### 2.6.2 Performance of InterHelper

In order to evaluate the performance of the fine-grained lane guidance application, we conducted 80 left and right turns at different intersections in Ann Arbor, Michigan, U.S., and the results are shown in Fig. 2.19.

The application is shown to be able to detect 95% of right turns with $R_1$, 90% with $R_2$, 90% of left turns with $R_3$ and 85% with $R_4$. We can therefore conclude that by integrating InterHelper into V-Sense, the application is capable of detecting lane changes in all

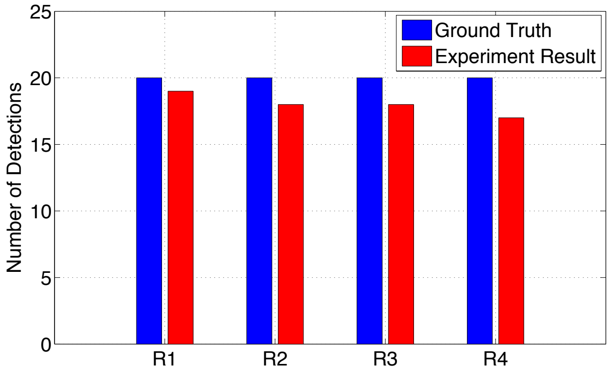


Figure 2.19: Performance of `InterHelper`.

cases, thus determining the vehicle's current lane.

## 2.7 Related Work

Detecting vehicle dynamics is critical for driving assistant systems and has also been an active research area. The related efforts can be categorized into two main approaches: camera-based or camera-free. For the camera-based approach, the vehicle or a stand-alone device detects the vehicle's maneuver by using its position (determined by the captured images) with respect to the lane boundaries [22, 31, 37]. Some commercial applications, such as iOnRoad [28] and Blacksensor [6], are capable of detecting lane departures by processing the images taken by cameras on smartphones, and thereby recognizes turns. However, all these methods require a mounted or built-in camera and also a clear view of the road. So, their performance can be seriously undermined if the visibility of the road is poor.

Camera-free systems achieve comparable (to camera-based systems) results regardless of the visibility of the road. The authors of [170] utilized the gyroscope, accelerometer readings of the smartphone and other direct readings from the OBD-II port to detect left/right turns and also the vehicle's real-time velocity. MIROAD [136] also utilized the gyroscope and accelerometer of the smartphone to acquire necessary data to detect vehicle motions via dynamic time wrapping (DTW). The authors of [108] utilized the orientation sensor and accelerometer to detect the driving pattern, thus determining if the driver is intoxicated. In contrast to [170], V-Sense is infrastructure-free, i.e., no additional hardware is required. It combines the GPS and IMU readings to acquire accurate and real-time velocity estimation. More importantly, in contrast with the existing work, V-Sense differentiates not only between left and right turns but also between lane change, U-turn, and driving on curvy roads. These in turn enable V-Sense to accurately handle various scenarios, such as fine-grained lane navigation in Section 6.

Furthermore, recent research and industrial efforts have been focusing on building driving assistant applications on smartphones. CarSafe [178] detects drowsiness by observing the driver's eye movement with the phone's front camera. Commercial applications [5, 6, 12, 28] utilized the phone's front camera to detect the front car and traffic lane, alerting the driver if any dangerous scenario is detected. Most of these applications require the user to mount the phone and use the camera to collect necessary information. This mount-before-go feature could limit the users' willingness, and eventually undermine the application's usability.

In contrast, V-Sense is a mount-free design for detecting steering maneuvers regardless of the position of the smartphone. Besides, without image processing, V-Sense incurs relatively lower computational cost. This mount-free design could pave s way for the development of many various driving assistant applications. For example, a recent study shows steering patterns could be utilized as a drowsiness indicator [41].

## 2.8 Conclusion

As an important and emerging subject in both research and industry communities, several driving assistant systems have been proposed. However, the capability of many existing work is limited by its reliance on the visibility of the road objects. Such an approach is only effective when the phone is carefully mounted and also has good visibility, i.e., its performance is undermined by its environment.

In this project, we proposed V-Sense, a camera-free middleware for driving assistant systems. V-Sense can accurately and inexpensively detect and differentiate vehicle steering by only utilizing built-in sensors on smartphones. By leveraging an effective bump detection algorithm and studying the nature of steering, V-Sense is capable of differentiating various steering patterns, such as lane change, turn, and driving on curvy roads. Based on the camera-free feature of V-Sense, we presented two proof-of-concept applications: careless steering detection and fine-grained lane guidance. V-Sense provides new functionalities without relying on cameras to provide a broader range of driving assistance.

# CHAPTER III

# Dri-Fi: Verify Drivers Using Turning-Behavior Biometrics

As transportation services, such as rideshare and on-demand delivery are migrating to mobile devices, impersonating a legitimate driver is emerging as a serious threat to the public. In this chapter, we first investigate two new types of attack related to this threat: *impostor* and *predator* attacks. By using a legitimate driver's account, the former allows ineligible drivers to bypass the background check, commonly enforced via one-time verification of legal documents when initializing the driver's account. In the latter, the fake driver simply approaches (e.g., trolling around late-night bars) careless passengers to commit crime(s). Such a fake driver has been very difficult, if not impossible, to trace. The most prevalent state-of-the-art countermeasure against these threats uses facial recognition, in which the service app asks the driver to take a selfie — a distractive, hard-to-use, and risk-prone practice. Moreover, with more obstructions such as face mask and PPE requirements during the COVID-19 pandemic, the traditional biometric check will become less practical.

We present `Dri-Fi` that enables commodity smartphones to accurately identify fake drivers by harvesting individuals' driving behavioral biometrics. `Dri-Fi` identifies drivers based solely on IMU sensors embedded in their smartphones. In particular, it captures and analyzes phone IMU data associated with the vehicle's turn(s). To characterize users' driving behavior, `Dri-Fi` extracts three new features from raw IMU data from their phones and feeds them to train a machine learning model. The design features of Dri-Fi enable *continuous* and *robust* detection of fake drivers using commodity smartphones without any restriction on device postures. Our extensive evaluation shows that Dri-Fi can differentiate drivers with the averaged balanced accuracy up to 96.3%.

## 3.1 Introduction

In recent years, there has been a rapid expansion of mobile transportation services, including rideshare (Uber, Lyft), on-demand delivery (Amazon, GrubHub, Door Dash, Uber Eats, Postmates), contactless rental cars (ZipCar, Maven), and auto insurance (Progressive, Metromile), just to name a few. By integrating these services with pervasive mobile platforms (smartphones), they have been reshaping the landscape of modern transportation service with unprecedented accessibility and flexibility.

As one of the top priorities, protecting the service and rider (in rideshare) from potential fraudulent and/or criminal acts has been enforced by examining drivers, such as background check which is required when a driver sets up his/her account. For example, rideshare companies commonly scrutinize their drivers and filter out ineligible drivers by checking their driver licenses, vehicle inspection reports, historical driving violations, and criminal records.

However, driven by financial interests and even criminal intentions [86], adversaries try to bypass the background check. According to several news reports [77, 79, 90], these fake drivers, if they successfully bypass the background check, can drive for transportation companies (e.g., cruise for passengers as a rideshare driver) and/or even commit crimes without timely notice by service providers and/or passengers. Unlike other common information technology loopholes such as fake social network account and password theft, this vulnerability can create even severer repercussions as it can cause not only financial harm but also physical violent crimes [86] like kidnap, rape, and robbery, to the users.

Next, we will present two unique threat models.

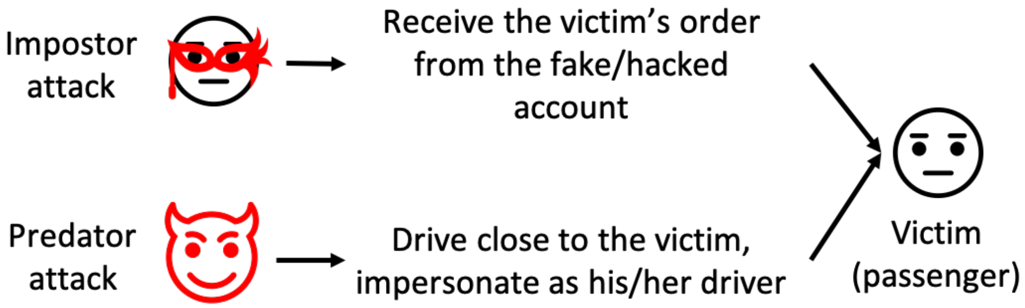

Impostor attack
Receive the victim's order from the fake/hacked account
Predator attack
Drive close to the victim, impersonate as his/her driver
Victim (passenger)

Figure 3.1: How impostor and predator approach the victim.

**Impostor attack.** The first step in an adversary's impostor attack is to get legitimate account(s). The adversary may borrow accounts from relatives/friends, rent/buy hacked accounts from underground markets or those are registered with compromised personal information [66]. The fake driver can then change the profile picture and vehicle license plate to further conceal the misconduct [79] . In practice, the adversary can receive the victim/passenger's order (as shown in Fig. 3.1) to achieve its goal, e.g., financial gains. This type of fraudulent behavior has already been reported to be active and has become a serious threat to the public. A recent investigation shows that a Uber account created (with compromised personal information) 27 days beforehand logged 233 trips across the Bay Area alone [85].

**Predator attack.** Targets of the predator attack are usually careless and vulnerable victims (passengers). According to existing reports [86], the adversary may troll around late-night bar and attract a victim's attention by impersonating (e.g., wave his/her hand) as the rideshare driver. Compared to the impostor attack, the predator attack usually has a criminal intention [86], and hence is more "vicious." According to a 2018 CNN report [82], 103 Uber drivers and 18 Lyft drivers had been accused of felony crimes, including sexual assaults or abuses.

Unfortunately, existing protection schemes (as we will elaborate in Sec. 4.2) fall short in detecting impostor and predator attacks. In particular, common biometric check methods such as face and/or fingerprint recognition distract driving and are thus infeasible for verifying drivers in real time. Their another limitation is usability. Amazon [81], Uber and Lyft prompt notifications and ask drivers to take selfies as a way to detect fraudulent drivers and/or criminals. But this has technical limitations related to photo quality, e.g., insufficient lighting when taking a picture. It is not continuous and hence may fail in practice — a fake driver could (1) spoof the authentication system with the *presentation* attack, in which the attacker uses artificial face or fingerprint replica [118, 161] to pass the examination, or (2) simply impersonate a legitimate driver between two check-ins.

To solve the fake driver problem for the fast-growing mobility app ecosystem, we propose `Dri-Fi`, a novel system for inferring the driver's identity based only on his/her smartphone's Inertial Measurement Unit (IMU) data. Specifically, `Dri-Fi` extracts the behavioral feature to capture the unique driving pattern of legitimate users. By feeding the feature to a machine learning pipeline, `Dri-Fi` can recognize any mismatch with the driver's behavioral pattern in real time, thus achieving *continuous* authentication.

The main challenge of `Dri-Fi` is how to derive a representative feature vector that represents the user's driving pattern by only using the smartphone IMU data. This is particularly challenging since the IMU only provides limited types of driving data, i.e., gyroscope and accelerometer. `Dri-Fi` overcomes this challenge by constructing the driving behavior profile based on *vehicle turns*, a common yet representative driving behavior. We choose vehicle turns to capture the driving behavior for two reasons. First, turns are behavior-rich actions that reflect how the driver accelerates/decelerates and how s/he steers. Second, turns are less likely to be affected by road and traffic conditions than other maneuvers. For example, a deceleration can be dictated by the preceding car, whereas turns are not. Once `Dri-Fi` detects a turn, it derives three new features that reflect the driver's driving behavior. As shown in Sec. 3.4, the feature vector is only affected by how the driver turns the steering wheel or how s/he presses the gas/brake pedal while making a turn. With the featuer vector, we can train the driver's model for authentication, even when s/he makes just *one* turn.

Based on a controlled study and large-scale field tests, we will later show (in Sec. 5.4) that these features vary only with drivers, but *not* with other factors such as smartphone models, car types, and trip routes, i.e., our driver feature vector is device-, car-, and route-*invariant*.

Our field study collected natural driving data from 30 different drivers with diverse backgrounds. Our results show that `Dri-Fi` can achieve the balanced detection accuracies up to 96.3%.

## 3.2 Motivation and Background

We first motivate `Dri-Fi` with the background information of the fake driver problem and the limitation of the state-of-the-art solutions. Then, we briefly introduce `Dri-Fi`'s unique approach to tackling the fake driver problem.

### 3.2.1 The Notorious Fake Driver Problem

As the cornerstone for protecting the passenger and/or delivery safety, mobility services have been using a strict background check to ensure enrolled drivers are legitimate and competent. However, impostor and predator attacks allow fake drivers to bypass this background check to impersonate a legitimate driver. The fake driver problem has already been reported to be rampant (Sec. 3.1). As the driver could be the transmission source of an infectious disease, this problem may also harm the public health in the case of a global pandemic like COVID-19 since it is very difficult, if not impossible, to trace fake drivers.

Mobility services have been actively enforcing their policies to tackle the fake driver problem. Specifically, these services forbid registered drivers from sharing their account with the un-registered [74, 81] to avoid untraceable accidents or criminal acts by impostors. However, account sharing is hard to prevent as the fake driver has the full access to the account.

An impostor can also purchase a compromised driver's account, which can be a hacked legitimate account [85]. Specifically, legitimate accounts are attractive targets of hackers [66] who can compromise and then monetize these accounts by selling them in the black market (e.g., dark web [62]). An adversary with strong malicious intent may use the predator attack that does not even require an account.

Mobility services including rideshare companies have been advocating tirelessly of safety tips [91], e.g., reminding the rider to ask the driver to exchange names, to mitigate fake driver problem. Unfortunately, people can easily overlook safety suggestions as there is no practical enforcement on following these tips. Hence, it is indispensable to have an automatic driver authentication method.

### 3.2.2 Existing Approaches

Biometric characteristics, such as face, retina, and fingerprint, are often used to authenticate users. However, these methods fall short in the driving context due the hard-to-use authentication process. For example, to use and pass the predominant selfie check on rideshare apps, drivers are strictly required to put the phone's front-facing camera in a certain distance from their faces, under a moderate light condition. Such strict usage requirements are also intrusive and even more risk-prone if an urgent check is enforced when the driver is driving. Moreover, it is obvious that biometric authentication is not practical for detecting the predator attack since the attacker does not even have an account.

During the evolving COVID-19 crisis, existing authentication methods are even more ill-suited for adapting to the new normal model and helping users reopen mobility services. Specifically, with stricter enforcement [92] of the mask, gloves, and other personal protection equipment (PPE), checking biometrics regularly would be even harder to use.

### 3.2.3 Behavioral-based Driver Authentication

Most state-of-the-art behavioral-based driver authentication schemes [99, 113, 127, 138, 141, 145, 149] require access to various data from in-vehicle networks, such as the

Controller Area Network (CAN), via an On-Board Diagnostic (OBD-II) dongle/device which must be plugged in the user's car. Specifically, researchers leverage various types (at least 48 types of data such as steering wheel angle, fuel consumption, etc.) of data generated by on-board ECUs (that are connected via in-vehicle networks) to extract representative features that capture people's driving behavior. There are two limitations in using this type of approaches in mobility apps:

- **Shutting down OBD-II ports.** Several recent vehicular hacks [59, 68] are making car-makers increasingly block/restrict in-car data access via the OBD-II port except when the vehicle is parked [69]. Thus, driver identification by accessing in-vehicle data will become less feasible.
- **Requirement of dedicated hardware.** They require dedicated devices (e.g., OBD-II dongles) for collecting the driving data, incurring an additional cost;
- **High overhead or difficulty of decoding in-vehicle data.** In-vehicle network messages are encoded by car-makers and their translation is proprietary. Therefore, unless the driver fingerprinting entity has access to such a translator, the messages must be reverse-engineered, which is painstaking and also incomplete.

Therefore, even when the mobility service provider has abundant resources to characterize the individual's driving pattern, it may still be very difficult, if not impossible, to overcome the above limitations.

### 3.2.4 Driver Authentication with `Dri-Fi`

`Dri-Fi` can be regarded as driver behavior biometrics (a.k.a. *implicit and continuous authentication* [154, 160]) by using phone sensors only. Specifically, `Dri-Fi` extracts intrinsic features directly from the phone's IMU sensor data (i.e., gyroscope, accelerometer) that captures distinct, measurable driving patterns. As we will present in Sec. 4.3 and further elaborate in Sec. 3.5, `Dri-Fi` can detect both impostor and predator attacks. It is worth noting that existing schemes like face recognition are unable to detect the predator attack as discussed in Sec. 3.2.2.

## 3.3 Overview of Dri-Fi

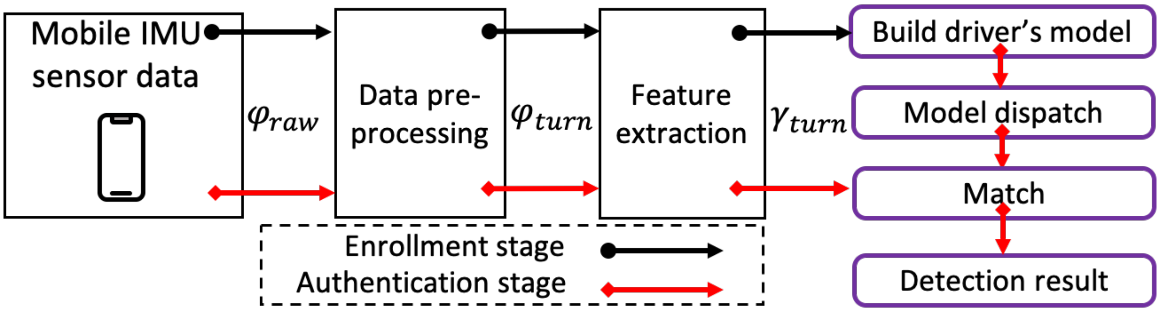


Figure 3.2: The system overview of Dri-Fi. The black and red lines presents data flows for enrollment and authentication stages, respectively.

The use of Dri-Fi requires enrollment and authentication. In the enrollment stage, Dri-Fi extracts the feature vector from the driver's smartphone data and builds the driver's behavioral model, which is associated with his/her account. The actual driver's presence at the enrollment stage can be enforced by a strict one-time facial check. Dri-Fi then enters the authentication stage: the driver's model and the extracted feature vector will be used to check if the current driving pattern matches with the driver's. In the design of Dri-Fi, only the model building step needs the communication with the back-end server. This is for keeping records and dispatching the trained model as we will introduce next.

Unlike many other authentication schemes, the design of Dri-Fi includes a unique *model dispatch module* to link the driver's model to the driver's account *and* the rider's account. This new design is essential for tackling different fake-driver (e.g., predator and impostor) attacks. As we will elaborate in Sec. 3.5, the authentication can be done on the driver's or the rider's phone to detect the impostor and predator attacks.

Fig. 3.2 depicts a detailed workflow of Dri-Fi as a 4-step process where Dri-Fi acquires the required raw IMU data $\varphi_{raw}$ for authenticating drivers. First, Dri-Fi pre-processes $\varphi_{raw}$ to remove noises and extracts the sensor measurements only while the driver was making a (left/right) turn, thus acquiring $\varphi_{turn}$ (Secs. 3.4.1 — 3.4.2). Next, based on the thus-obtained $\varphi_{turn}$, Dri-Fi constructs a set of features (Sec. 3.4.3), i.e., a feature vector $\gamma_{turn}$. The feature vector can be used for training the driver's model or real-time authentication.

## 3.4 Characterizing Turning Maneuvers

In this section, we will elaborate the system design of data pre-processing and feature extraction.

### 3.4.1 Data Pre-processing

During a trip, Dri-Fi continuously collects the raw IMU data $\varphi_{raw} = \{gyro_{raw}, acc_{raw}, mag_{raw}\}$ from the phone's gyroscope, accelerometer, and magnetometer, respectively.

#### 3.4.1.1 Data Calibration

To handle different postures of the mobile device inside a car, Dri-Fi aligns the coordinate of IMU readings using the magnetometer [102]. Specifically, Dri-Fi aligns the device's coordinate with the geo-frame/earth coordinate so as to maintain the consistency of analysis. This protects the data which Dri-Fi uses for driver authenticating from the effects of the IMU's pose, thus *unneeding* a fixed device's posture.

#### 3.4.1.2 Data Pre-processing

Once the coordinate-aligned data of the gyroscope and accelerometer sensors have been collected, Dri-Fi smoothes and trims them to prepare for further analyses. If the device which Dri-Fi uses is a smartphone, its handling by the user may cause high-power noises on the gyroscope and accelerometer sensors. Abnormal road conditions (e.g., potholes) may also incur a similar level of noise. Therefore, Dri-Fi first filters out abnormal spikes in the data. Dri-Fi then smoothes each IMU sensor (gyroscope and accelerometer) data stream by using a low-pass filter to remove high-frequency (with the cutoff frequency set to 20Hz) noises. This can help mitigate the impact of different car models, as the high-frequency noises are usually induced by the varying vibration pattern [135] of different car models and road excitations.

### 3.4.2 Extraction of Left/Right Turns

Dri-Fi trims the smoothed data further by retaining the IMU measurements acquired only during a left/right turn, $\varphi_{turn} = \{gyro_{turn}, acc_{turn}, mag_{turn}\}$, i.e., smoothed IMU data of only left/right turning maneuvers. In other words, measurements taken when the driver constantly drove on a straight road, or when the car stopped to wait for traffic lights or stop signs are all discarded. Among the various maneuvers (e.g., turns, lane changes, acceleration/deceleration), the reason for Dri-Fi's focus on data from turns is that the vehicle/driver's turns are less affected by the car in front (i.e., traffic condition) than others. For example, deceleration of a vehicle would depend on the car in front, whereas left/right turns are less likely to depend on it.

In order to extract only data related to left/right turns, Dri-Fi uses the (coordinate-aligned) gyroscope's yaw rate reading as it reflects the vehicle's angular velocity around its vertical axis, i.e., the vehicle's rotational inertia. Note, however, that non-zero gyroscope readings do not necessarily represent a left/right turn, since there exist other (similar) maneuvers such as lane changes and U-turns which incur similar results [102]. So, from the gyroscope, Dri-Fi extracts data of only left/right turns in the following two steps:

**S1.** Recognizes whether or not a steering maneuver — which we refer to as *maneuvers* (left/right turns, lane changes, U-turn, etc.) that suddenly change the vehicle's heading direction significantly — was made;

**S2.** Determines whether the steering maneuver was a left/right turn and, if yes, extracts

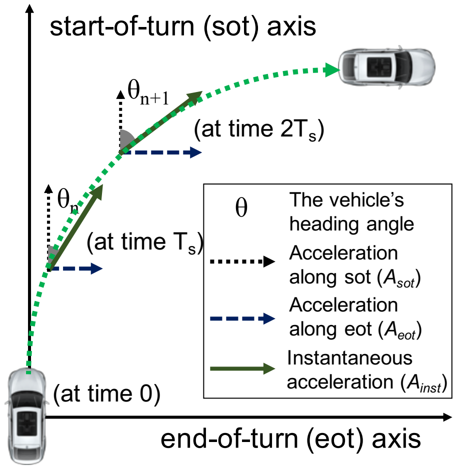

start-of-turn (sot) axis
$\theta_{n+1}$
(at time 2$T_s$)
$\theta_n$
(at time $T_s$)
(at time 0)
end-of-turn (eot) axis
θ The vehicle's heading angle
Acceleration along sot ($A_{sot}$)
Acceleration along eot ($A_{eot}$)
Instantaneous acceleration ($A_{inst}$)

Figure 3.3: Accelerations and headings of a turn.

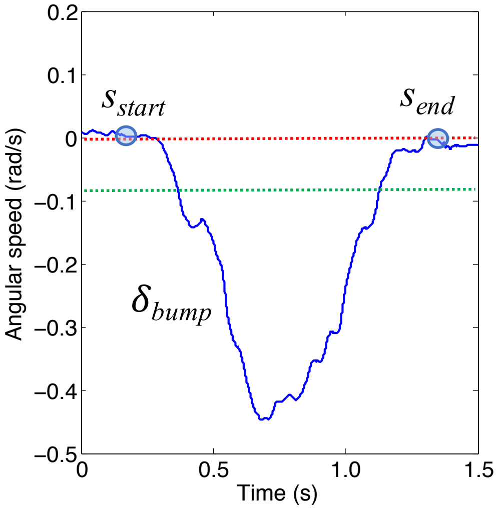


Figure 3.4: Turn extraction from gyroscope readings.

the sensor readings acquired during that turn.

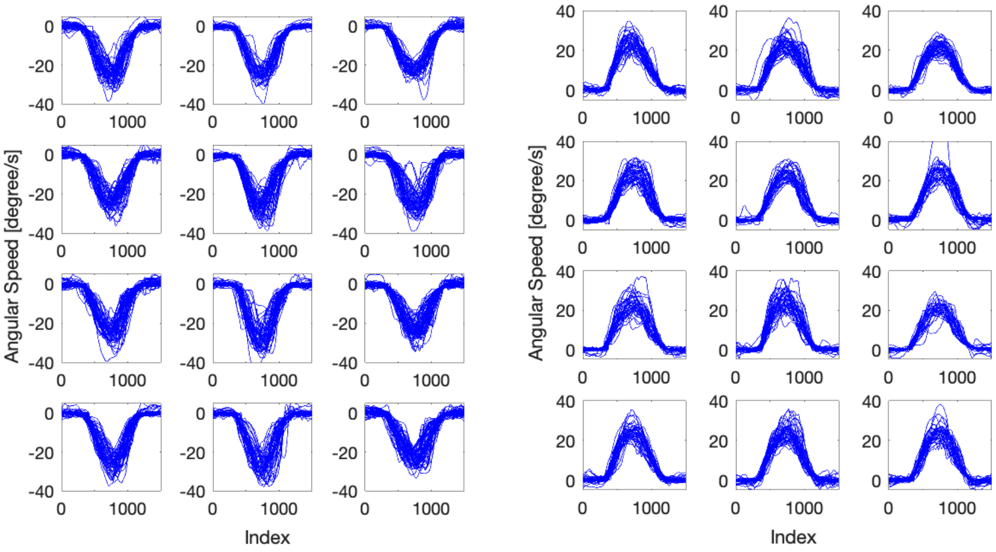
**S1. Recognizing steering maneuvers.** `Dri-Fi` recognizes the occurrence of a steering maneuver when the yaw rate readings from the gyroscope form a "bump-shape". When a car changes its direction by turning left, as shown in Fig. 3.4, the yaw rate reading from the gyroscope first decreases, then reaches its minimum peak, and finally rises back to approximately 0 rad/s upon completion of the left turn. For a right turn, everything would be the opposite to a left turn; increase, reach the maximum, and decrease. Depending on how the coordinates are aligned, a negative (positive) bump may reflect a right (left) turn. However, in this paper, we consider the yaw rate to increase when rotated clock-wise. Based on this observation, `Dri-Fi` determines that a steering maneuver has occurred if the absolute yaw rate exceeds a certain threshold, $\delta_{bump}$, which is empirically set to 0.15 $rad/s$. Note that without the threshold ($\delta_{bump}$), even a small movement of the steering wheel would cause `Dri-Fi` to mis-detect a steering maneuver. Thus, `Dri-Fi` marks the start time/point of that steering maneuver as $s_{start}$ when the absolute yaw rate, $|Y|$, exceeded $\delta_{bump}$ for the first time. Also, `Dri-Fi` marks the end point, $s_{end}$, when $|Y|$ first drops back below $\delta_{bump}$. Since the steering would in fact have started a bit before $s_{start}$ and ended a bit later than $s_{end}$, where $|Y| \approx 0$ as shown in Fig. 3.4, `Dri-Fi` moves points $s_{start}$ and $s_{end}$ backwards and forwards, respectively, until $|Y| \approx 0$. As a result, Dri-Fi interprets a steering maneuver to have made within the interval $s = [s_{start}, s_{end}]$.

**S2. Filtering left/right turns.** The steering maneuver extracted in S1 may be comprised of not only left/right turns but also lane changes or U-turns, since those maneuvers yield similar bump-shaped yaw rate readings. In order to extract only left/right turns, as in [102], Dri-Fi derives the change in the vehicle's heading angle, which is defined as the difference in the vehicle's heading angle between the start and the end of a steering maneuver. Fig. 3.3 shows an example vehicle trajectory during a right turn where three IMU sensor readings were acquired at times $t = s_{start} + \{T_s, 2T_s, 3T_s\}$, i.e., sampled with frequency of $1/T_s$. As in step S1, let $t = s_{start}$ be the time when the vehicle was detected to have started the turn. Since the yaw rate readings from the gyroscope represent the vehicle's angular velocity around the vertical (Z) axis, the change in the vehicle's heading angle after time $nT_s$ has elapsed since $s_{start}$ is:

$$\theta[nT_s] \approx \theta[(n-1)T_s] + Y_n T_s = \sum_{k=1}^{n} Y_k T_s \tag{3.1}$$

Here $Y_n$ denotes the $n$-th yaw rate reading since $t = s_{start}$. Therefore, at the end of making a right turn, the eventual change in the vehicle's heading angle, $\theta_{final} = \theta[s_{end} - s_{start}]$ would be approximately 90° whereas at the end of a left turn it would be -90°. This change in the vehicle's heading angle is a good indicator for determining whether the vehicle has made a left/right turn, since for lane changes, $\theta_{final} \approx 0°$, whereas for U-turns, $\theta_{final} \approx 180°$. Thus, Dri-Fi calculates the $\theta_{final}$ of a detected steering maneuver (made during $s_{start} \sim s_{end}$), and only retains it such that $70° \leq |\theta_{final}| \leq 110°$, i.e., approximately ±90°. Note that since left/right turns usually take a few seconds (<3 seconds), the drift in the gyroscope reading during a turn [151] does not affect Dri-Fi's performance.

As a result, whenever the driver makes a left/right turn, Dri-Fi can acquire a sensor data streams (i.e., gyroscope and accelerometer readings) which are outputted only during



(a) Left turn.

(b) Right turn.

Figure 3.5: Interpolated gyroscope readings of left and right turns from 12 different drivers.

the turn, i.e., during $s = [s_{start}, s_{end}]$. However, since different road geometries may result in different turning radii, the length of the readings may vary, which may affect `Dri-Fi`'s performance. Thus, in order to make `Dri-Fi`'s authenticating accuracy independent of path selection and dependent on the driver only, we interpolate the sensor data stream into a fixed length. This also facilitates `Dri-Fi` to authenticate the driver even when using two different devices that may have different sampling rates.

The effect of the interpolation step is shown in Fig. 3.5. Each subplot presents the gyroscope readings of left/right turns performed (then extracted with our scheme) by one of 12 different drivers. In summary, we have 682 left turns (mean: 56.8, std: 5.5) and 511 right turns (mean: 42.6, std: 5.4); we will later in Sec. 3.6 elaborate on the collection of this data. From Fig. 3.5, we gain two insights: (1) the near-equivalent shapes of gyroscope readings indicate that via interpolation, the analyses can be done from a consistent vantage point, despite turns made in different environments (e.g., road geometries); and (2) the time-series data (e.g., gyroscope readings) of left and right turns shows a clear morphological difference, i.e., convex vs. concave. Therefore, it is necessary to build different classifiers for left and right turns. Our experimental results (in Sec. 3.7.1) indicate that both left and right turns classifiers have similar performance. This can be explained from the near-equivalent time durations of left and right turns — the mean and standard deviation for left turns are 1.18 and 0.095 seconds, whereas those for right turns are 1.12 and 0.101 seconds. Thus, one can use either the left or right-turn classifier as they have similar performance in identifying/verifying drivers. So, we focus on using left turns for identifying/verifying drivers. As the analysis pipeline is transferable, one may easily adapt `Dri-Fi` based on right turns.

### 3.4.3 Formulating the Feature Vector

Whenever the driver makes a left/right turn, `Dri-Fi` acquires an IMU sensor data stream $\varphi_{turn}$. The main challenge in authenticating the driver is determining which features to extract from the data stream.

#### 3.4.3.1 Feature Extraction

`Dri-Fi`'s feature extraction stems from both prior work and our observation of real-world driving patterns. Prior work on driver identification [113, 180] has shown that statistical features derived from the acceleration and vehicle steering (e.g., angular speed) can be useful for characterizing driving patterns. We have also observed that the turning is a *behavior-rich* maneuver and reflects how each driver accelerates/decelerates and steers a car. Moreover, turns are less likely to be affected by road and traffic conditions than other maneuvers. We, therefore, extract the following three new features from the filtered IMU sensor data for driver authenticating:

$F_1$. Acceleration along the end-of-turn axis ($A_{eot}$);

$F_2$. Deviation of $F_1$ ($\Delta A_{eot}$); and

$F_3$. Deviation of the raw yaw rate ($\Delta Y_{raw}$).

As depicted in Fig. 3.3, we define the start-of-turn (SOT) axis as the axis/direction in which the vehicle was detected to have started its turn (direction at time $s_{start}$). In reference to the SOT axis, we define the *end-of-turn* (EOT) axis as the one orthogonal to the SOT axis. That is, regardless of the change in the vehicle's heading angle after the turn (e.g., 95° for a right turn), by definition, the EOT axis is set perpendicular to the SOT axis.

**F$_1$. Acceleration along the EOT axis.** The acceleration along the EOT axis is an interesting yet powerful feature in Dri-Fi since it represents *both* 1) how much the driver turns his/her steering wheel and 2) *at that moment* how hard the driver presses the brake/acceleration pedal during the left/right turn. In other words, it reflects one's (unique) turning style. We will later show through extensive evaluations that the features we use for Dri-Fi do *not* depend on the vehicle type or route but only on the driver's unique maneuvering style. Note that instantaneous acceleration, which we refer to as the acceleration along the vehicle's heading axis, measured during a turn would only reflect the driver's input/actions on the brake/acceleration pedal but not on the steering wheel. Similarly, the instantaneous yaw rate, i.e., the angular velocity of the vehicle, measured from the gyroscope would only reflect the driver's steering actions.

For deriving the vehicle's *acceleration along the EOT axis* when $nT_s$ seconds has elapsed since $s_{start}$, $A_{eot}[nT_s]$, Dri-Fi utilizes the vehicle's instantaneous acceleration, $A[nT_s]$, at that moment (obtained from the accelerometer) and its change in the heading angle, $\theta[nT_s]$ (extracted from the gyroscope) as

$$A_{eot}[nT_s] = A[nT_s]sin(\theta[nT_s]) \tag{3.2}$$

In addition to the acceleration along the EOT axis, the value along the SOT axis may also be used. However, since the information Dri-Fi would obtain from the accelerations along the SOT axis will be redundant when those along the EOT axis are already available, we do not consider them as features in Dri-Fi; this also reduces the feature space.

As an alternative to $A_{eot}$, one can think of using centripetal/lateral acceleration, which would be perpendicular to the vehicle's instantaneous acceleration ($A$). However, since the

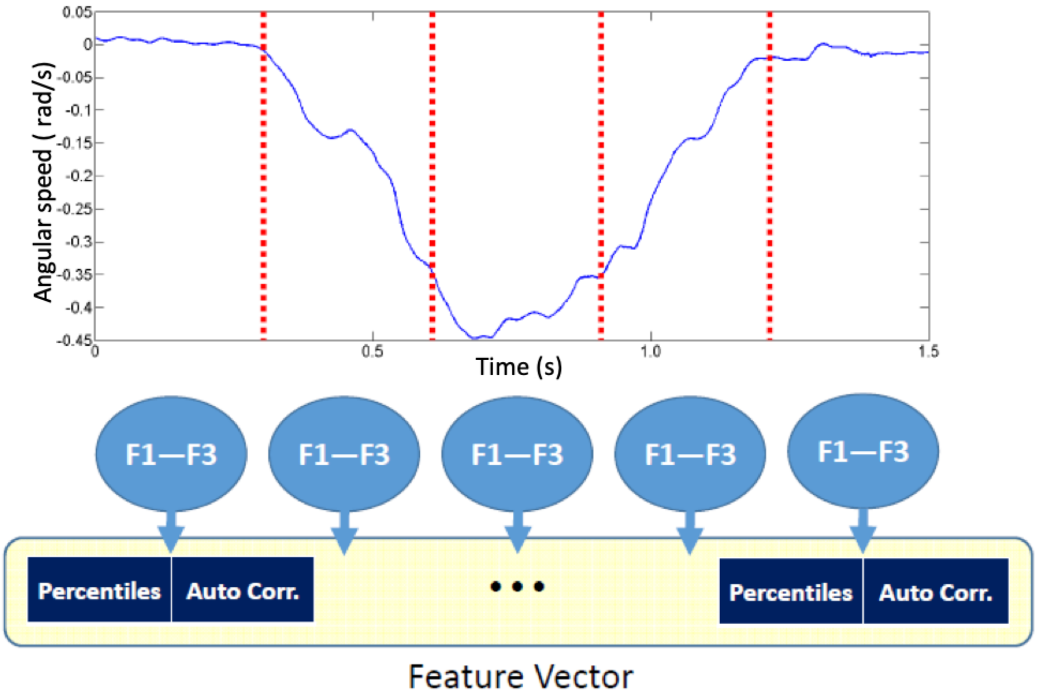


Figure 3.6: `Dri-Fi`'s construction of feature vector.

centripetal acceleration is affected by the turning radius (depends on road geometry, car size, and smartphone placement), whereas the acceleration along the end-of-turn axis is not, we do not consider this for features in `Dri-Fi`.

**F$_2$–F$_3$. Deviations of $A_{eot}$ and raw yaw rate.** `Dri-Fi` derives not only $A_{eot}$ but also $\Delta A_{eot}$, i.e., difference between the subsequent acceleration values along the EOT axis. Since $\Delta A_{eot}$ reflects how *aggressively* the driver concurrently changes his steering and pedal actions during a turn, this feature captures the driver's aggressiveness.

In addition to $\Delta A_{eot}$, `Dri-Fi` also determines the deviations in the *raw* yaw rate measurements, $\Delta Y_{raw}$. Note that in order to accurately extract turns, `Dri-Fi` pre-processed the data with a low-pass filter. However, as the turns are already extracted, in order to not lose the accurate understanding/interpretation of how aggressively the driver turns his steering wheel, `Dri-Fi` also derives $\Delta Y_{raw}$; the driver's aggressiveness shown from the low-pass filtered data would have been reflected in $F_2$. In addition to the driver's aggressiveness of turning the steering wheel, this feature also captures how stable the

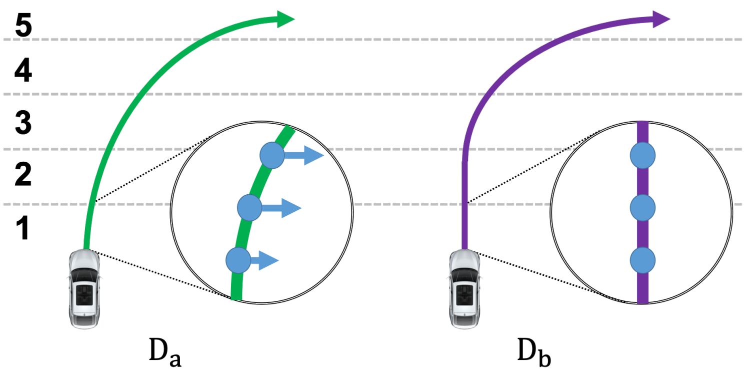


Figure 3.7: Different autocorrelations depending on the driver's turning style.

driver maintains an angle during the turn(s) and thus helps Dri-Fi's driver authenticating.

#### 3.4.3.2 Feature Vector Construction

To construct the feature vector $\gamma_{turn}$ for classification and thus authenticating, Dri-Fi transforms $F_1$–$F_3$ as follows:

1. Upon detection of a turn, as shown in Fig. 3.6, Dri-Fi divides the IMU measurements (acquired during the turn) into 5 stages, each with an identical duration.
2. For each stage, Dri-Fi determines $F_1$ — $F_3$.
3. For each of $F_1$—$F_3$, Dri-Fi selects its $\{10, 25, 50, 75, 90\}$-th percentiles and autocorrelations at 1–10 lags due to their effectiveness evidenced by our experiment. Then, Dri-Fi aggregates the metrics for constructing a feature vector.

Note that Dri-Fi generates an instance with such a feature vector per (detected) turn. With the percentiles, Dri-Fi understands the distributions of $F_1$—$F_3$ in each stage of turn.

Meanwhile, a more powerful feature for Dri-Fi in authenticating the driver is the *autocorrelations* of $F_1$—$F_3$ in each stage of turns. Fig. 3.7 shows an example of two different drivers making a right turn. When making the turn, one can see that $D_a$ started turning his steering wheel during stage 1 of the turn whereas $D_b$ started it later in stage 3. As shown in Fig. 3.7, which also illustrates the accelerations along the EOT axis ($A_{eot}$) during stage 1, one can see that an early turn from $D_a$ incurs non-zero values of $A_{eot}$ in stage 1 of the turn. On the other hand, since $D_b$ drives further on a straight line along the SOT axis, his $A_{eot}$ values in stage 1 would approximately be 0. Similarly, values of $F_2$ and $F_3$ would also remain 0 for $D_b$, but not for $D_a$. As a result, the autocorrelations of $F_1$—$F_3$ for $D_a$ would show significantly different values from those for $D_b$, i.e., drivers' different turning styles lead to different $F_1$—$F_3$ autocorrelations.

Then, are these autocorrelation values of $F_1$—$F_3$ different enough between drivers to be considered as a driver's behavioral signature? Also, for a given driver, are those values consistent across multiple left/right turns? Fig. 3.8 shows the boxplots of $F_1$ autocorrelations for two drivers — who participated in our evaluations — during their first stage of left turns. We will later elaborate on the evaluation settings in Sec. 5.4. One can see that since the *tendencies* of drivers moving straight or turning the steering wheel early/late at the early stages of turns were different, the autocorrelations (at different lags) between the two drivers were obviously distinguishable. Moreover, one can see that although the driver was making those left turns at different times and places, the variances in some autocorrelation lags were quite low, i.e., stable. Not only the first stage but also stages 2$\sim$5 showed a similar distinctiveness and stability. This shows that the autocorrelations of $F_1$—$F_3$ are not only distinct among drivers but also quite stable for a given driver, i.e., drivers' turning styles are relatively constant and distinct, so as to function as the core for `Dri-Fi` in authenticating the drivers.

Using the constructed feature vector $\gamma_{turn}$ as an input data for training machine classifiers (e.g., Random Forest [153]), `Dri-Fi` can authenticate the driver as soon as the driver has made only one (left/right) turn.

## 3.5 Detection of Impostor & Predetor Attacks

Now we elaborate the defense mechanism against impostor and predator attacks. The workflow of `Dri-Fi` authentication is shown in Fig. 3.9. After a rider requested a ride, the authentication flow starts by checking if the driver has confirmed the pickup on his/her

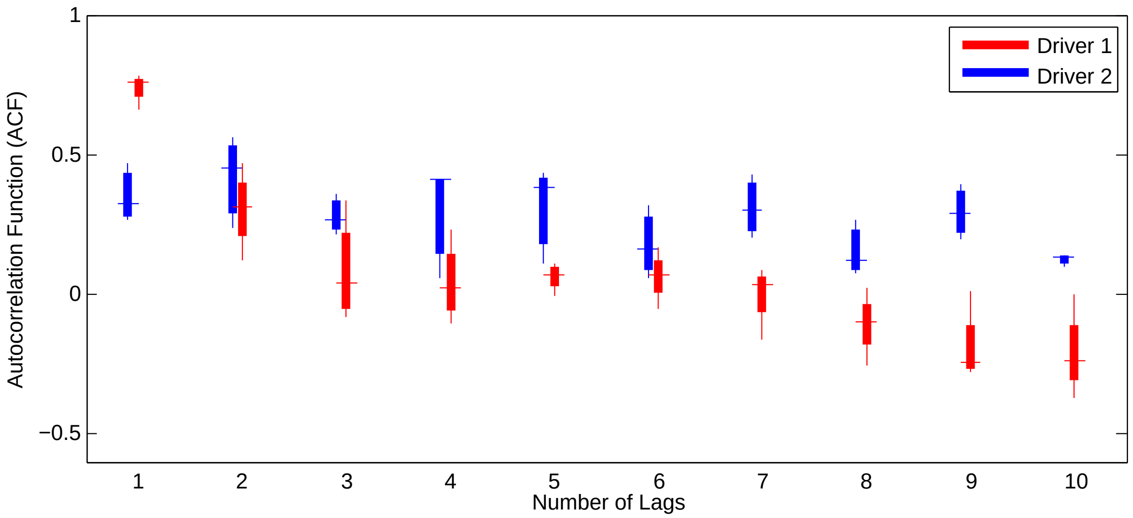


Figure 3.8: Correlogram of feature $F_1$ for two drivers.

app. Note that a predator cannot confirm this since s/he is not the assigned driver or may not even have an account.

**Detect an impostor attack.** If the driver confirmed the pickup, `Dri-Fi` starts on the *driver*'s device to check if the behavioral pattern matches the enrolled pattern. If a mismatch is detected, a highly-suspicious impostor report will be generated and sent to the service provider.

**Detect predator attack.** If the driver has not yet confirmed the pickup, the rider is either looking/waiting for the ride or already boarded a car, i.e., a suspicious predator attack. To verify this, the workflow first analyzes if the moving speed of the rider's phone (detected by using the GPS speed) exceeds a certain threshold, denoted as $V_{TH}$. A reasonable indicator of a moving car in a city can be around 30 mph [72]. If yes, `Dri-Fi` starts on the *rider's device* to check if the current behavioral pattern matches the enrolled pattern of the designated driver. If a mismatch is detected, the rider's phone generates a predator alert to alarm the rider, and a report of this incident will also be sent to the service provider.

Further contextualization and/or action associated with the report can be taken by the service provider, e.g., calling local law enforcement immediately and/or suspending the

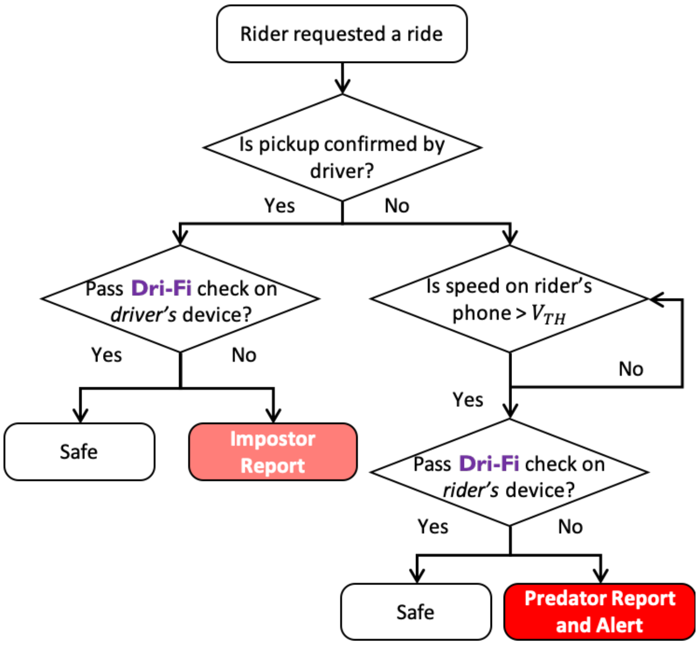

Rider requested a ride
Is pickup confirmed by driver?
Yes
No
Pass Dri-Fi check on *driver's* device?
Yes
No
Safe
Impostor Report
Is speed on rider's phone > $V_{TH}$
No
Yes
Pass Dri-Fi check on *rider's* device?
Yes
No
Safe
Predator Report and Alert

Figure 3.9: The workflow of `Dri-Fi`'s authentication process.

impostor's account. Note that this workflow can be slightly modified for other mobility services, such as on-demand delivery. For example, if the delivery driver is an impostor, the authentication process can start once the driver receives a delivery request.

To validate `Dri-Fi`'s efficacy in detecting both types of attack, it is essential to first analyze its performance and accuracy in capturing behavioral patterns, which are elaborated in Secs. 3.7.1 — 3.7.4. Since the riders are likely to place the phone in a back seat of the car, we study how to deal with varying phone posture.

The key challenge in achieving accurate driver verification is how to mitigate the impact of the varying phone *posture*. For example, in the case of an impostor attack, the phone is usually mounted at or next to the driver's seat when a driver's account is being used by the impostor, whereas in the case of predator attack, the phone is usually located in the back seat with the rider. Our experiments imitate the user's placement of the smartphone. As shown in Fig. 3.10, the reference point indicates the windshield mount — the predominant placement by the driver (i.e., for the impostor attack scenario). Points 1, 2, 3, and 4 are popular placement by the rider (i.e., for the predator attack scenario) left backseat pocket,

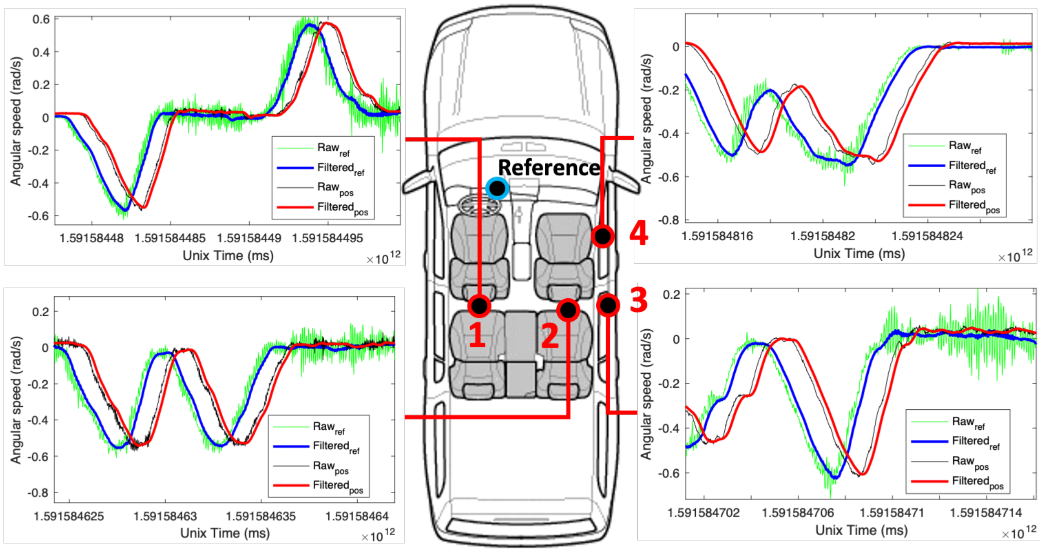


Figure 3.10: The impact of different placement on the raw and filtered sensor readings.

right backseat pocket, right rear door handle, and right front door handle, respectively. We placed one Google Pixel phone in the reference point and another Pixel phone in other points for four short trips. The goal is to study the difference of the sensor reading induced by device placement.

According to vehicle moving dynamics [170], different positions on car have same angular speed, thus enabling a position-invariance feature $F_3$ As shown in the plots in Fig. 3.10, the gyroscope data collected from two different phones, i.e., Redmi Note8 (running Android Pi) and Galaxy S5 (running Android Marshmallow) shows identical morphological patterns. Note that, these two data traces are not perfectly synchronized: on average Galaxy S5 phone's data is 0.104 seconds delay compared with Redmi Note8. This is because two phones' different hardware configuration and clock systems. As for feature $F_1$ and $F_2$, our end-of-turn acceleration also remove the impact of the turning radius, which may vary due to the road geometry and phone placement. Hence, the design of Dri-Fi should be capable for addressing of the varying-posture issue. We will show further experimental results in Sec. 5.4.

## 3.6 Setup and Design of Experiments

The goal of our experimental evaluation is to answer the following two questions.

Q1. Can `Dri-Fi`'s feature vector accurately capture the driver's behavioral characteristics, or is the feature vector invariant with other impact factors, i.e., device model, device posture, car type, and route?

Q2. Does `Dri-Fi` address the driver masquerading attack in a real-world setting?

We answer these questions by first presenting the data-collection methodology, and then elaborating on the construction of our dataset.

### 3.6.1 Data Collection

We implemented the data collection front-end app on Android smartphones. As we elaborated in Sec. 4.3, the data collection app records the IMU sensor data (during driving), which is subsequently uploaded to the server for further analysis (in Sec. 5.4). To test the app's performance across different platforms, we installed the app on five different Android smartphone models (i.e., Google Pixel, Redmi Note 8, Nexus 5X, Samsung Galaxy S5, and Samsung Note 5) which support API level from 23 to 28. We asked the drivers to turn on the data-collection app before each trip and upload the data after completing the trip. Note that we do *not* require the smartphone to be placed at a single fixed location. We instead recommended our participants to avoid moving their smartphones too much/frequently, which may change their phones' postures significantly/frequently.

### 3.6.2 Design of Experiment

We collected the driving data in two phases, i.e., controlled study and large-scale test. We recruited 30 drivers from our university campus for this experiment. Subjects were assigned to different driving tasks to meet the design principles of our two-phase study. Since our system does not require any personal information from the users, the Institutional Review Board (IRB) of our university classified this effort as non-regulated.

#### 3.6.2.1 Phase 1. Controlled Study

To understand if Dri-Fi's feature is invariant of other factors, i.e., car and route, our experiment objects are comprised by three sets, i.e., drivers, cars, and routes. The driver set ($\mathbb{D}$) including five different drivers recruited from our university, with average age at 29.2 (STD: 5.6). Here $\mathbb{D} = \{d_1, d_2, d_3, d_4, d_5\}$. The car set $\mathbb{C}$ consists of three different cars, denoted as $\mathbb{C} = \{c_1, c_2, c_3\}$. Our vehicle selection includes a wide range of car types, $c_1$, $c_2$, and $c_3$ are Ford Explorer (full-size SUV, weight 2.1 ton), Lincoln MKZ (mid-size sedan, weight 1.6 ton), and Toyota Corolla (compact-size sedan, 1.3 ton), respectively. The posture set $\mathbb{P} = \{p_1, p_2, p_3\}$ includes three representative placements, namely, windshield mount, frontseat cupholder, and backseat door handle for $p_1$, $p_2$, and $p_3$, respectively. Note that, we observed that smartphone's orientation is not always 90° upright (i.e., the y-axis of the device is perpendicular to the ground) in all postures. e.g., the device may tilted some degrees when it placed in $p_2$ and/or $p_3$. Our coordinate alignment process (Sec. 3.4.1), if works properly, should calibrate the sloping position. Finally, the route set has three different routes in a suburban area in the U.S. Here, $\mathbb{R} = \{r_1, r_2, r_3\}$, with average route length at 3.4 miles (STD: 0.6). There are 15, 20, and 18 left turns and 18, 16, 14 right turns for $r_1$, $r_2$, and $r_3$, respectively.

| Test | Change Factor | Fixed Factor | Subjects |
|---|---|---|---|
| $T_1$ | Driver | Car, Posture, Route | $\mathbb{D}$, $p_1$, $c_1$, $r_1$ |
| $T_2$ | Driver, Car | Posture, Route | $\mathbb{D}$, $\mathbb{C}$, $p_1$, $r_1$ |
| $T_3$ | Driver, Posture | Car, Route | $\mathbb{D}$, $c_1$, $\mathbb{P}$, $r_1$ |
| $T_4$ | Driver, Route | Car, Posture | $\mathbb{D}$, $c_1$, $p_1$, $\mathbb{R}$ |

Table 3.1: Summary of evaluations.

By controlling different factors, i.e., driver, car, and route, we designed three tests (as shown in Table 3.1):

- *Pilot test ($T_1$)*: Here we test driver verification's performance with fixed car and route. We asked all drivers to drive $c_1$ on $r_1$, which resulted in $|\mathbb{D}|$ trips.
- *Car-invariant test ($T_2$)*: We asked all participants to drive all cars on route $r_1$. This

test gives us $|\mathbb{D}| \times |\mathbb{C}|$ trips.

- *Posture-invariant test ($T_3$)*: We asked all participants to place their phones on three different positions as indicated in $\mathbb{P}$ while driving $c_1$ on route $r_1$. This test gives us $|\mathbb{D}| \times |\mathbb{P}|$ trips.
- *Route-invariant test ($T_4$)*: We finally test the impact of route choices by asking all drivers to drive $c_1$ on all routes. This gives us $|\mathbb{D}| \times |\mathbb{R}|$ trips.

As the simplest test, $T_1$ allows us to have a initial insight of our feature's performance (see Sec. 3.7.1) on capturing behavioral pattern. $T_2$ helps us learn the impact of behavioral pattern (Sec. 3.7.2) amid changing vehicles. We also studied the consistency of our feature when the user changes his/her car, e.g., using the trained pattern on $c_1$ for verifying a driver when he/she drives $c_2$. Similary, we studied in $T_3$ on the impact of smartphone posture. In $T_4$, we tested the feature's efficacy and consistency amid changing routes in Sec. 3.7.4.

#### 3.6.2.2 Phase 2: Large-scale Study

We further enlarged the number of drivers by adding 25 new participants to conduct a stress test of `Dri-Fi`. Due to the larger search space, `Dri-Fi` may show degraded performance in verifying drivers. Different from Phase 1, no factor is controlled in Phase 2. All experiments were conducted in a mid-sized city in the Midwest of the U.S. In total, our dataset includes 30 (21 male and 9 female) drivers with an age span of 22–50; we collected data from 25 cars of 8 different models. The timespan is over 6 months (between 9AM and 10PM) which consists of more than 201 hours of driving in urban/suburban areas and covers trips of 3,198 miles. On average, each driver produced driving data for approximately six hours and 106.6 miles. According to our survey after data collection, none of the participants indicated that our data collection app affected their normal driving. Hence, this experiment recorded each driver's natural driving data.

With the large-scale test, we are able to (1) examine the feature's performance under real-world settings without constraints, and (2) analyze the feature consistency with respect to changing observation set and errorneous training data. Results will be shown in Sec. 3.7.5.

## 3.7 Evaluation

We first show the evaluation results in Phase 1 and 2. Then, we present the overhead of `Dri-Fi` app. Finally, we demonstrate how `Dri-Fi` detects the impostor and predator attacks in practice.

### 3.7.1 Pilot Study Result

We examine both left and right turn classifiers' performance in our pilot study. Based on the data collected from $T_1$, we trained a 100-tree Random Forest (RF) classifier for both left turns and right turns, respectively. We use 80% of the turns and leave the remaining 20% as the test set. To obtain an accurate estimate of the model prediction performance, we used 10-fold cross validation.

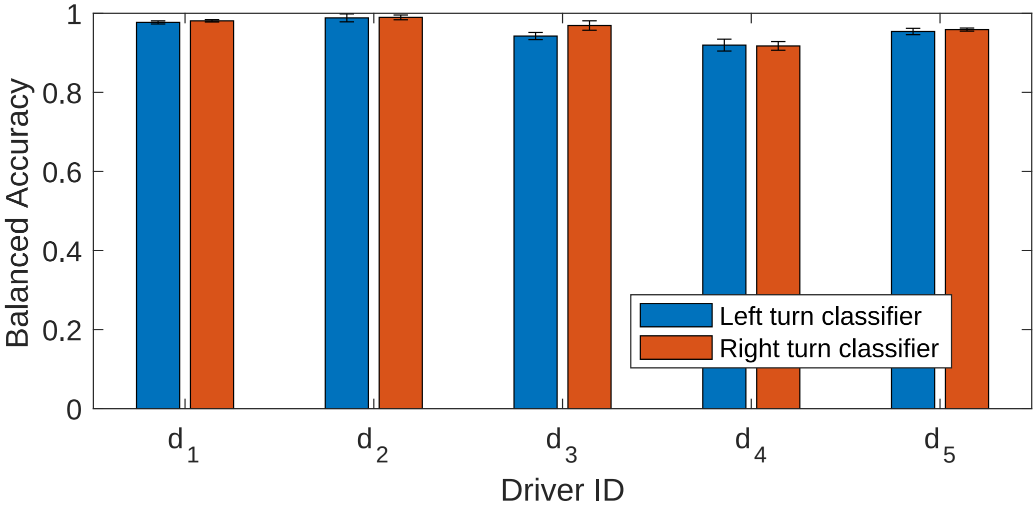


Figure 3.11: Performance of the pilot study.

The averaged balanced accuracies (i.e., the average of recall obtained on each class [89]) are 0.9563 (averaged STD: 0.0092) and 0.9632 (averaged STD: 0.0072) for left turns and right turns, respectively. That is, `Dri-Fi`'s user (e.g., app developer) can use either left or right turn classifier. The experimental results show that our feature vector can accurately capture the behavioral pattern.

### 3.7.2 Car-invariance

The effect of car choices may come from two sources: (1) the high-frequency noises induced by mechanical vibrations from the car chassis, and (2) possible behavioral change for adapting to a different car. Our algorithm design filters out (Sec. 3.4.1) the high-frequency noises. In $T_2$, we focus on investigating the behavioral change amid different car choices. For each car model, we use the corresponding data (from $T_2$) to train the driving model of each driver. Then we use all car choices' data for testing the model. For example, for $c_1$, the training subject set is $\{\mathbb{D}, c_1\}$; the testing subject sets include $\{\mathbb{D}, c_1\}$, $\{\mathbb{D}, c_2\}$, and $\{\mathbb{D}, c_3\}$. Similar to the pilot study, we applied 10-fold cross validation and obtained the overall averaged balanced accuracy as shown in Fig. 3.12.

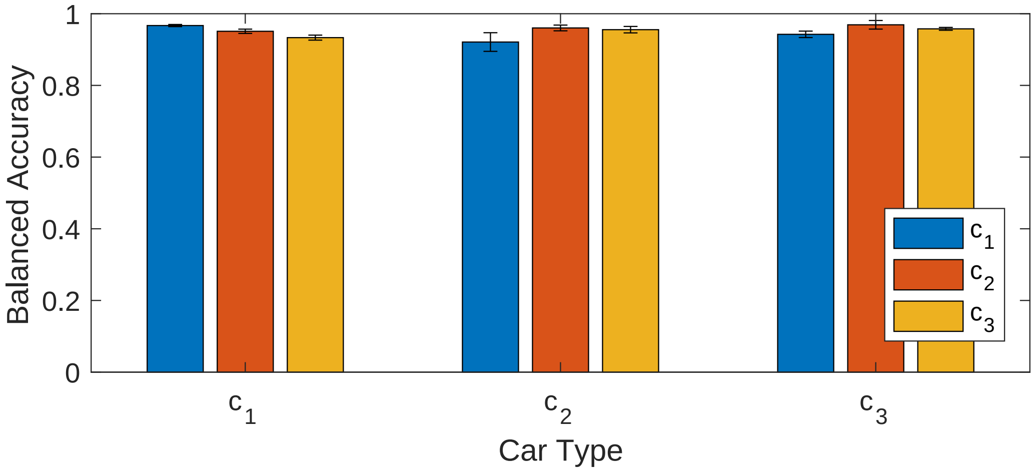


Figure 3.12: The impact of changing cars.

As shown in Fig. 3.14, the overall accuracy and STD (combined and denoted as $\{mean, STD\}$) for cars $c_1$, $c_2$, and $c_3$ are $\{0.9504, 0.0053\}$, $\{0.9456, 0.0143\}$, and $\{0.9565, 0.0083\}$, respectively.

### 3.7.3 Posture-invariance

As discussed in Sec. 3.5, for different postures, after addressing the variation in vibration pattern, the key differences are induced by time delay and marginal longitudinal velocity. By conducting $T_4$, we assess the impact of postures with a similar method and the 10-fold cross validation as stated in Sec. 3.7.2. For each posture, we use its corresponding data to train the model for every driver, and then use all postures' data for testing the model. Note that, if the training and testing sets are from the same trip, we leave 80% data for training and 20% data for testing.

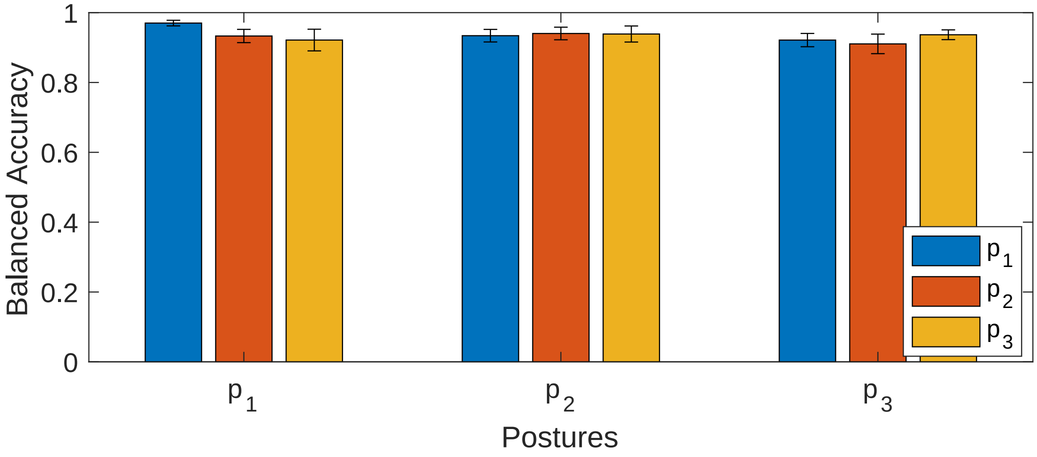


Figure 3.13: The impact of different postures.

As shown in Fig. 3.14, the overall accuracy and STD for route $p_1$, $p_2$, and $p_3$ are $\{0.9416, 0.0193\}$, $\{0.9377, 0.0197\}$, and $\{0.9229, 0.0203\}$, respectively. Note that for $p_1$, the performance shows a relatively large degradation compared with $p_2$ and $p_3$. Our speculation is, there were some movements in steering maneuvers due to the loose fixture of the phone in $p_2$ and $p_3$. These movements induces low-frequency noises that undermine the classifier performance.

### 3.7.4 Route-invariance

The effect of the driving route may come from three sources: (1) varying high-frequency noises due to different road pavement conditions, (2) varying traffic condition, and (3) different intersection geometries. To address the last two sources, as emphasized in Sec. 3.4.1, we apply trimming and interpolation steps. By conducting $T_4$, we assess the effect of route selection with a similar method stated in Sec. 3.7.2. For each route, we use its corresponding data to register the model for every driver, and then use all routes' data for testing the model. Again, a 10-fold cross validation and averaged balanced accuracy are used.

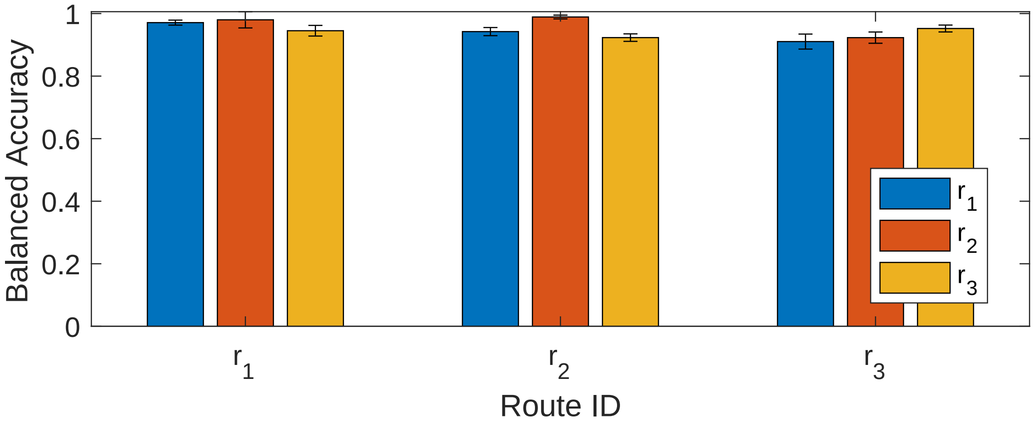


Figure 3.14: The impact of different routes.

As shown in Fig. 3.14, the overall accuracy and STD for route $r_1$, $r_2$, and $r_3$ are $\{0.9654, 0.0170\}$, $\{0.9515, 0.0103\}$, and $\{0.9285, 0.0177\}$, respectively.

After validating Dri-Fi's consistency amid varying car choices, route selections, and varying posture we now proceed to a larger-scale test of Dri-Fi's performance.

### 3.7.5 Large-scale Test

In this phase, we evaluated Dri-Fi's performance in classifying up to 30 drivers (25 new drivers plus the 5 drivers in the first two phases) using the data collected *without* control factors. All participants (including three couples) drove their own cars (six Corolla and nine Camry) for the experiment. We focus on evaluating Dri-Fi's performance in terms of changing training data size, time-lapse consistency, and resistance against erroneous training data.

#### 3.7.5.1 Observation Set

To emulate an increasing search space, we examined the averaged balanced accuracies of all drivers in this test. We apply 10-fold cross validation on the test. As shown in Fig. 3.15, the overall trend indicates that an expanding observation set, e.g., with an increasing number of drivers, can help the classifier to better handle diversities, thus achieving better classification performance.

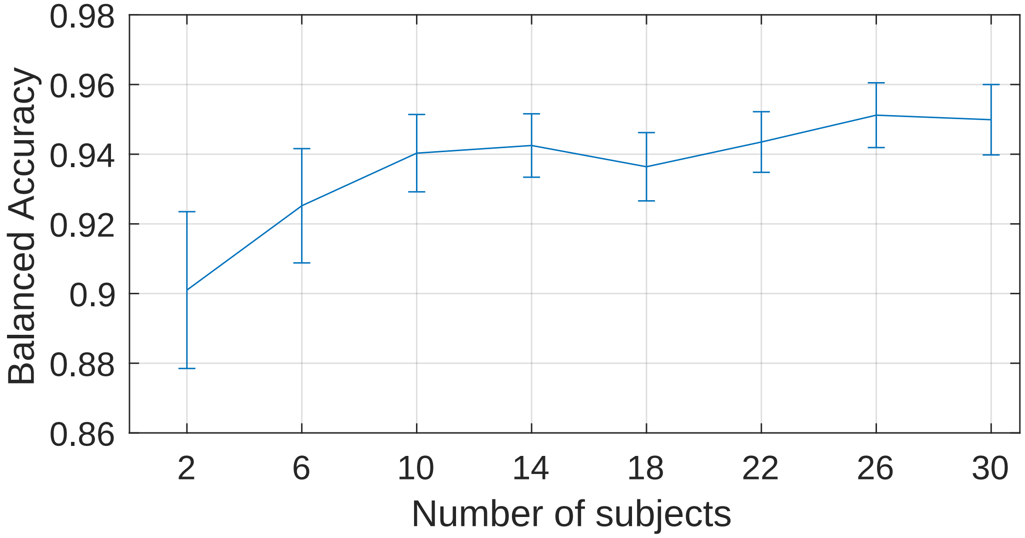


Figure 3.15: Performance with the increasing observation set.

#### 3.7.5.2 Erroneous Training Data

we emulate disturbances due to sudden behavioral changes, by applying incorrect labels while constructing the training set. Specifically, a turn was made by $d_1$ but we deliberately label it as a negative sample (i.e., made by another driver). We arbitrarily picked and labeled some turns by *any* of the 30 drivers. The number of arbitrarily picked turns with erroneous labels was varied via parameter $p_{err}$, which represents the percentage of such erroneous labels.

Fig. 3.16 shows `Dri-Fi`'s performance with $p_{err}$=0∼20%. Even when the training dataset for `Dri-Fi` contains 20% of erroneous labels, `Dri-Fi` can still achieve 0.8717

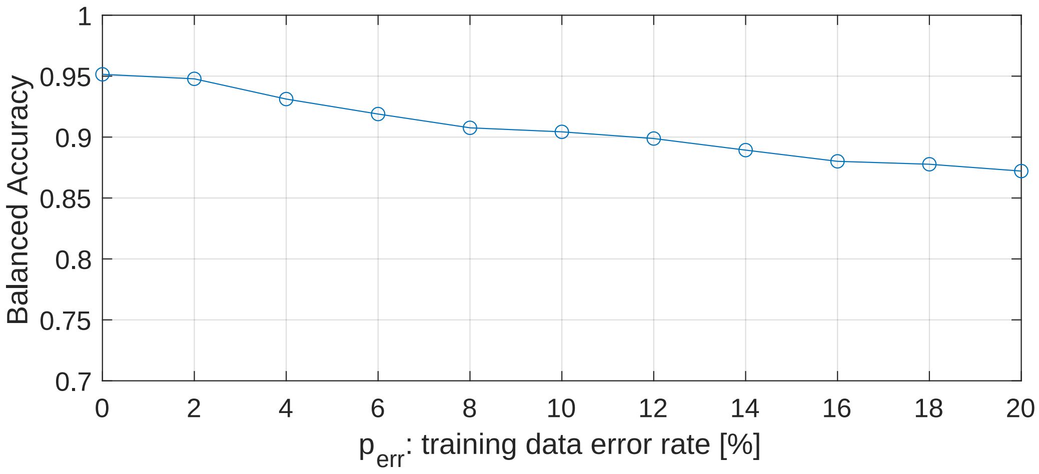


Figure 3.16: Classification accuracy with $p_{err}$ % erroneous training dataset.

balanced accuracy accuracy with only one turn. That is, even when the training data is imperfect, Dri-Fi can still achieve good verification accuracy, and is thus robust to training errors.

### 3.7.6 User Study

So far, we have evaluated Dri-Fi's performance with real-world driving data while varying the control factors. To better understand Dri-Fi's user experience, we implement it in a fully-functioning smartphone app and demonstrate how it works in practice.

#### 3.7.6.1 App Development and Experimental setting

We first implemented a demo app of Dri-Fi on Android. To embed the machine learning pipeline atop of the data collection module in a smartphone (as introduced in Sec. 3.6), we implemented an embedded machine learning (embedded ML) pipeline by using Java predictive model markup language (PMML) Android library [84]. We then serialized the user's driving model into a binary file (in the .pmml.ser format) an average size of which is only 906 KB with STD of 113 KB. The model dispatch process (as discussed in Sec. 4.3) can be achieved by sending the serialized file to the driver's/rider's smartphone.

We collected feedbacks from two participants in our driver pool on both the enrollment and authentication stages of `Dri-Fi`. For the enrollment stage, they are asked to first drive their cars for five minuets with their phone mounted on the windshield. For the authentication stage, they can place their phones at different spots (as shown in Fig. 3.17).

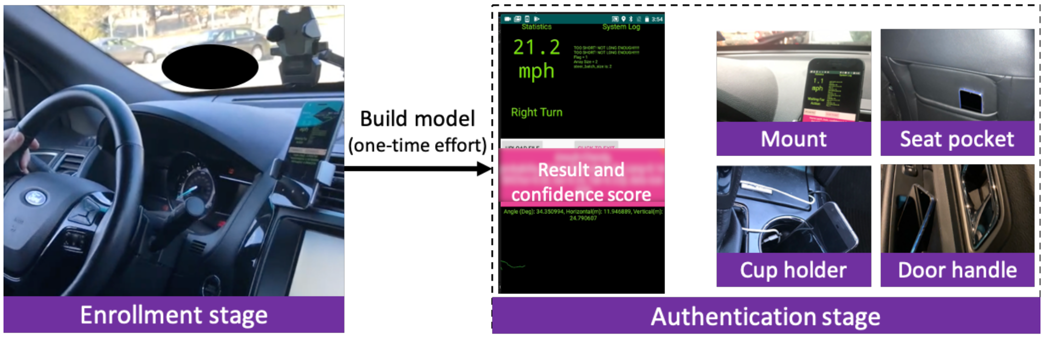

[redacted]

Figure 3.17: Illustration of `Dri-Fi` app. Training and usage phases have different phone postures.

#### 3.7.6.2 Usability Comparison: `Dri-Fi` vs. FaceID

We also surveyed users for their experience with `Dri-Fi` as compared to FaceID [88], which is the state-of-the-art facial biometric authentication method embedded in the most recent high-end iPhones. Specifically, we investigated users' experience in both enrollment and authentication stages.

**Enrollment stage.** For setting up the user's model, FaceID requires sufficient light and minimal facial obstructions. In our experiment, the participants complained about the difficulty in setting up their face model when they are wearing face masks (the smartphone shows a "Face Obstructed" alert) during the current COVID-19 pandemic. In contrast, the participants found the *five-minute* driving for training `Dri-Fi` is more acceptable because of its less intrusive nature.

**Authentication stage.** In this stage, `Dri-Fi` shows two key advantages: (1) it can authenticate the driver when the phone is placed in the backseat, i.e., the common seat for riders, and (2) it achieves continuous authentication, which is convenient for the driver to use.

### 3.7.7 System Performance of Dri-Fi

We have evaluated Dri-Fi's overhead — i.e., classification time delay, CPU usage and energy consumption — on mobile devices for its data collector and embedded ML modes. Despite the common process of steering recognition and IMU sensor usage, different working modes have their *extra* real-time workloads. Specifically, the collector mode requires real-time I/O on the local data file; whereas the embedded ML app needs to call the JPMML library for analyzing a data batch in real time. We didn't evaluate the network consumption, e.g., interaction with the server, as it is a one-time effort designed by the specific app.

We recorded the CPU usage on Google Pixel (1.6GHz quad-core CPU) and LG Nexus 5X phones (hexa-core CPU with four 1.4GHz Cortex-A53 and two 1.8GHz Cortex-A57) by using the Android Developer Bridge (ADB) shell. To profile Dri-Fi's battery consumption, we used Google's Battery Historian tool [122], which allows developers to inspect the battery usage of each app/module from the bug report generated by smartphones. Note that on average, drivers in the US drive 50.6 minutes and 31.5 miles per day [164]. Hence, we generate the bug report from smartphones with 50-min usage of Dri-Fi. To emulate the normal number of turns that a driver may make, we steer the phone to generate 10 left turns and 10 right turns (each turn last 3 seconds). To evaluate the extra overhead incurred by Dri-Fi's embedded ML module, which requires an extra overhead in calculating the classification result, we compared the CPU usage and battery drain of Dri-Fi in the data collection mode and the embedded ML mode.

We report the performance (time delay, CPU usage, and battery consumption) in Table 3.2. The extra time delays introduced by the embedded ML are 2.4 and 3.1 seconds on Pixel and Nexus 5X, respectively. The results also indicate that the classification process in embedded ML mode incurs more CPU overhead, i.e., 2.61% (1.72x) and 3.56%

| Model | Metric | Data Collector | Embedded ML |
|---|---|---|---|
| Pixel | Delay | N/A | 2.4 s |
| | CPU | 1.52% | 4.13% |
| | Battery | 0.90% | 1.34% |
| Nexus 5X | Delay | N/A | 3.1 s |
| | CPU | 1.66% | 5.22% |
| | Battery | 1.12% | 2.33% |

Table 3.2: The averaged time delay, CPU usage, and battery usage of Dri-Fi in different working modes.

(2.14x) more CPU usage on Pixel and Nexus 5X, respectively. We also see a lower increase of battery usage: 0.44% (0.49x) and 1.21% (1.08x) extra battery consumption on Pixel and Nexus 5X, respectively. Overall, the extra cost of Dri-Fi should be acceptable to mobility app users.

## 3.8 Related Work

State-of-the-art driver authentication schemes use various in-vehicle sensor data to identify the driver [110,113,127,141,145,149,169]. Enev *et al.* [113] investigated whether one can authenticate the driver via in-vehicle CAN data. By exploiting 18 or more types of CAN sensor data (e.g., brake pedal position, throttle position) collected through the OBD-II port for at least 15 minutes, it was shown that the driver can be identified. Similarly, the feasibility of driver authenticating based on CAN data was shown in [110, 138, 146, 147, 169]. Corbett *et al.* [106] investigated the performance of using both CAN and smartphone data for formulating driver's behavioral profile. Kwak *et al.* [141] also exploited CAN data for driver authenticating for anti-theft.

Hallac *et al.* [127] exploited 12 different types of CAN data for driver authentication. Specifically, they proposed a classification algorithm by exploiting simple to complex features such as mean, standard deviation, and spectral components of the 12 different CAN sensor data. That way, they were able to identify the driver with high accuracy within one turn.

Van Ly *et al.* [145] also used in-car CAN data representing acceleration, brake, and turn signals to identify the driver. Other relevant work includes [149] which used not only CAN data but also additional new features including the car-following distance and the sound information when someone speaks inside the car for driver identification.

These related studies have demonstrated the feasibility of driver authentication, but *all* of them are based on in-car data. Accessing and interpreting/decoding CAN data require are non-trivial problems and require additional hardware. Wijnands *et al.* [172] used neural network for formulating a user's driving behavior by using the acceleration derived from GPS data alone. However, their approach requires months of data in order to learn the user's driving pattern.

Riener *et al.* [159] designed a seat mat with embedded pressure sensor array to characterize different drivers' sitting postures. Although this approach can also achieve continuous driver identification, installation of the dedicated mat could be too high an overhead for mobility apps.

Recently, researchers have also explored ways of identifying drivers based on mobile sensor data [162, 180]. Zhang *et al.* [180] proposed the use of both IMU and in-vehicle sensors for classifying a small number of drivers with high accuracy. Tahmasbi *et al.* [162] described a system for differentiating drivers with smartphones' IMUs. They collected natural driving data from three couples who share a car. However, their approach required strict smartphone placement and did not evaluate the effect of varying control factors.

Compared to existing approaches, `Dri-Fi` presents a more thorough study and has several distinct advantages in authenticating drivers based solely on IMU readings. In particular, `Dri-Fi` achieves driver authentication with high accuracy as soon as the driver makes a single turn. Its accuracy improves significantly as the driver makes more turns.

## 3.9 Limitation and Scalability of `Dri-Fi`

We now discuss the limitation and other potential use-cases of `Dri-Fi`.

### 3.9.1 Limitation

With the coordinate alignment discussed in Sec. 3.4.1, Dri-Fi can analyze the data from the geo-frame coordinate irrespective of the device's posture. However, if the device constantly moves during driving (e.g., the driver's smartwatch), it may introduce noise, thus causing Dri-Fi to be less accurate. To overcome such a limitation, Dri-Fi can be configured to process the data only when clear turn patterns are obtained (e.g., a clear bump-shaped gyroscope measurement as in Fig. 3.4). However, this might make Dri-Fi acquire 'meaningful' turns data less frequently and thus affect its overall performance, especially when only a few turns are made within a trip. Moreover, depending on the driver's emotion, sense of urgency, and physical status, Dri-Fi's accuracy may vary as well. So, in future we would like to conduct a detailed study of the effects of such factors on Dri-Fi's accuracy.

It is also possible that the driver's behavior may exhibit a drastic change due to other factors such as serious mental disturbance, emergency, and intoxication. It is worth noting that driving under these influences, from the safety perspective, should be reported to the service provider and/or the rider; the driver should also stop temporarily accepting driving assignments.

### 3.9.2 Scalability

Dri-Fi has the potential for benefiting other mobility apps.

**Auto insurance.** Verifying the driver's identity is essential for detecting fraudulent acts in the fast-emerging pay-as-you-drive (PAYD) auto insurance [63, 71, 75, 87]. Current PAYD is susceptible to the policyholders fraudulent acts. That is, the policyholders of PAYD may fool the system to reduce their insurance premium by lending the account to drivers (e.g., close friends, relatives) to get a safe driver rating. Such "benign" driver-impersonating problems also exist in car rental businesses. Dri-Fi can be integrated into the mobile app of these services to detect such fraudulent acts.

**Anti-theft.** Beside its anti-fraud detection for businesses, `Dri-Fi` also has potential for helping individual users prevent/detect car theft. The required sensing and computational modules of `Dri-Fi` can be met by commodity smartphones, and/or streamlined into an embedded device such as an energy-efficient concealable IoT device, which can be instrumented in the car. The anti-theft system enabled by `Dri-Fi` has a unique advantage by harvesting the behavioral biometrics check — it can alert (e.g., sending an SMS message to) the vehicle owner when the user's driving behavior does not match the recorded/expected pattern.

To this end, `Dri-Fi` can be the primary means of continuous, real-time authentication. It can also work as a user-friendly secondary authentication that supplements existing biometrics for better usability. In future, we will explore more usage models of `Dri-Fi`.

## 3.10 Conclusion

We have presented `Dri-Fi`, a driving data analytic framework for verifying drivers based on their maneuvering behavior biometrics. `Dri-Fi` achieves this by capturing new representative features of a driver's unique way of making turns. Via extensive evaluations, `Dri-Fi`'s extracted features are shown to represent the driver's unique turning style and thus function as the key in differentiating drivers. `Dri-Fi`'s efficacy and usability are essential for addressing novel attack factors, i.e., impostor and predator attacks, as we introduced in this paper. `Dri-Fi` renders various use-cases in today's mobility ecosystem. Moreover, as the world is reopening after the COVID-19 pandemic, mobile-centric transportation services will remain as an essential part of our lives and businesses.

# CHAPTER IV

# TurnsMap: Uncovering Unprotected Left Turns via Crowdsensing

Left turns are known to be one of the most dangerous driving maneuvers[1]. An effective way to mitigate this safety risk is to install a left-turn enforcement — e.g., a protected left-turn signal or all-way stop signs — at every turn that preserves a traffic phase exclusively for left turns. Although this protection scheme can significantly increase the driving safety, information on whether or not a road segment (e.g., intersection) has such a setting is not yet available to the public and navigation systems. This chapter presents a system, called `TurnsMap`, that exploits mobile crowdsensing and deep learning to classify the protection settings of left turns. One of our key findings is that crowdsensed IMU sensor (i.e., gyroscope and accelerometer) data from onboard mobile devices can be used to recognize different types of left-turn protection. `TurnsMap` first collects IMU sensor data from mobile devices or smartphones carried by the driver/passenger(s) in a moving car. It then feeds the data to an analytics engine powered by (1) a data mining engine for extracting and clustering left turns by processing raw IMU data, and (2) a deep-learning pipeline for learning the model from the IMU data to identify the protection type of each left turn. We have built and used a large-scale real-world driving dataset to evaluate `TurnsMap`, demonstrating its capability of identifying different left-turn enforcements

[1] In this work, we focus on right-hand traffic countries. Our approach can be applied for right turns in left-hand traffic countries.

with 90.3% accuracy. A wide range of automotive apps can benefit (e.g., enhancing traffic safety) from the left-turns information unearthed by `TurnsMap`.

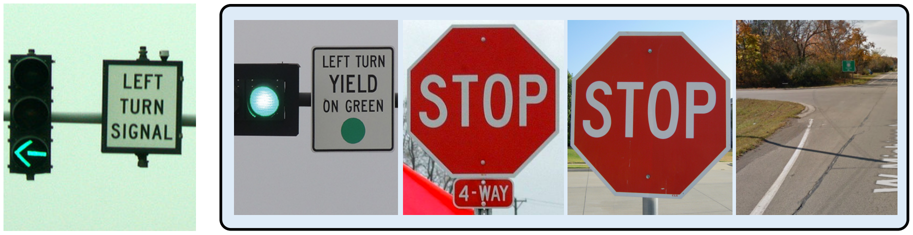
## 4.1 Introduction

Left turns are one of the most dangerous driving maneuvers that account for a large percentage of fatal traffic accidents [38]. For example, among the different driving maneuvers at intersections, left-turns are reported to be the deadliest due to oncoming (interrupting) traffic and pedestrians crossing the street the car is turning onto. According to a survey report by the U.S. Department of Transportation, left-turns-related accidents alone constitute *53%* of all intersection-related crashes in the U.S. [38]. In fact, to mitigate left-turns-related accidents, some companies like UPS are enforcing their delivery trucks to avoid, if possible, all left turns [49].

There have been extensive transportation engineering efforts to enhance the safety of left turns by deploying different left-turn regulations at intersections with high-frequency left turns. In general, these regulations can be categorized as protected and unprotected settings.

**Protected settings.** At a protected left-turn, when the left-turn signal (e.g., a green left arrow as shown in Fig. 4.1(a)) is turned on, only left-turning vehicles are permitted. This is the *most* effective way to avoid/reduce left-turn-related accidents. According to the NHTSA report [36], an intersection that has a protected left-turn signal can effectively reduce the left-turns-related accidents by **87%**, making the intersection safer for all traffic. Based on the traffic regulation [26], a left turn protection (e.g., a green leftward arrow in the U.S.) reserves a *traffic phase* — a time period assigned to certain traffic movements — exclusively for left turns.

**Unprotected settings.** At an unprotected left-turn, when vehicles are permitted to make left turns, it is the drivers' responsibility to stay alert and avoid interrupting



(a) Protected

(b) Unprotected

Figure 4.1: Different left-turn settings.

traffic and/or pedestrians. For example, in Fig. 4.1(b), circular traffic lights (i.e., traffic lights that only show round-shaped light colors) and stop signs (as shown in the first three figures in Fig. 4.1(b)) provide less restrictive protection by halting the crossing traffic (i.e., perpendicular to the ego-vehicle's initial heading). The rightmost figure in Fig. 4.1(b) shows the scenario in which no sign/signal is installed to regulate traffic. Drivers of left-turning vehicles have to be extremely careful of, and avoid interrupting traffic/pedestrians from all possible directions.

The information of left-turn settings is essential for drivers to enhance safety by using it *before* making dangerous left turns; for example, drivers can choose to avoid less-protected left turns when planning a trip route with Google maps.

Unfortunately, such information is not yet available to the public/drivers for two reasons.

- *Lacking public datasets.* Although traffic signals and signs are installed by the local transportation department, the corresponding specification is fragmented or not even digitized due to the diverse development levels for different regions. For example, Waze's left-turn avoidance function [18] is available only in major cities of California.
- *High cost and low update-rate of road surveys.* Information of traffic lights/signs can be extracted from the images of intersections captured by a limited number of heavily instrumented road survey vehicles (e.g., mapping cars for Google StreetView). However, updating the road information is hampered by the prohibitive

road-survey cost. As a result, updating the database of on-road images may take years even in well-developed regions. These limitations prevent a large coverage of road survey services and timely reflection of changes of traffic lights/signs.

To narrow this gap and evaluate left-turns safety by discovering and sharing the left-turn setting information on road segments (e.g., intersections), we have developed `TurnsMap`, a novel system that can effectively differentiate protected/unprotected left turns by using the crowdsensed data (i.e., gyroscope, accelerometer, and GPS data) collected from mobile devices carried by drivers and passengers. In particular, `TurnsMap` is empowered by mobile crowdsensing and a novel data analytics pipeline. It does not require any additional hardware from the users but harvests the sensor data which are already being generated by various apps (e.g., fitness and health apps) on users' mobile devices. The low overhead/cost of `TurnsMap` facilitates its timely and economical deployment.

`TurnsMap`'s data analytics scheme dives deep into the real-world driving behavior. Specifically, given a large number of instances (e.g., extracted from crowdsensed data) of left turns, for *less* protected left-turns (i.e., the protected setting is stricter than the unprotected ones), the interruptions caused by on-coming vehicles and/or pedestrians are more likely to occur. To quantify and analyze this characteristic, `TurnsMap` is powered by two key components. The first component is a *data mining engine*; after collecting the crowdsensed data, `TurnsMap` extracts left turns by analyzing the raw sensor data. `TurnsMap` then clusters left turns to find left-turn *hotspots* (road segments that have high-frequency left turns. This concept will be elaborated in Sec. 4.5). The second component is a *deep learning pipeline* (Sec. 4.6) that can learn the features of the underlying intersection information from the crowdsensed data. We use a novel data augmentation to enhance the size and representativeness of the dataset, which can be used for training with a recurrent neural network based on using long-short term memory (LSTM) units.

The evaluation of `TurnsMap` is also challenging due to the lack of mobile sensing-based dataset collected from drivers and ground truths of left-turns information. The mobile driving data is collected from smartphones of 18 different drivers (14 males and 4 females), building a 1.6GB dataset over a cumulated travel distance of more than 3,589km. To collect ground truths, we have designed an interactive traffic-scene annotation system (Sec. 4.6). We then recruited 231 participants on Amazon Mechanical Turk for the annotation task. Finally, we have aggregated the annotations from the participants to construct a new dataset which is comprised of 965 labeled left-turn hotspots. Our evaluation results show that `TurnsMap` is able to differentiate left-turn settings with 90.3% accuracy.

Our proposed system makes the following main contributions:

- Building and demonstration of `TurnsMap`, a mobile crowdsensing system for classifying protected/unprotected left turns;
- Design, implementation and evaluation of a novel data analytics pipeline for large-scale time-series data classifications.

## 4.2 Motivation

We motivate this work by answering two questions: (1) how does `TurnsMap`'s revelation of left-turns information actually help the automotive ecosystem? and (2) what would be the advantage(s) of using mobile sensing?

### 4.2.1 How Does Classifying Left-Turn Settings Enhance Driving Safety?

As discussed in Sec. 5.1, the protected left-turn setting are the most effective way to mitigate the safety risk at intersections. However, it is not feasible or even desirable to install the protection at every left-turn hotspot, because strict protection of left turns (Fig. 4.1(a)) degrades the overall traffic throughput. For example, since a protected left-turn signal only allows left-turning vehicles to go through the intersection, NHTSA suggests such a signal to be installed only at those intersections where all intersecting roads have a similar traffic volume [30]. Since drivers (especially novice drivers with limited driving experience) may choose only protected turn lefts, the information of protection settings of left turns, if publicly available, will help them and automotive apps to enhance safety.

**Navigation Systems** As we will elaborate in Sec. 4.8, planning a route with a minimum number of high-risk (i.e., un-/less-protected) left turns in navigation systems (e.g., Google maps , Waze [18]) by utilizing left-turns enforcement information would be highly desirable. Let us consider a typical usage scenario. If the user enables this functionality on his/her navigation app, then it can exploit the left-turns information on all possible routes to generate the route with the least number of risky left-turns while ensuring an acceptable *estimated arrival time* (ETA) at the destination. Note that by providing its users with safer travel paths, a navigation app can also efficiently collect data from its users to help expand/update `TurnsMap`, achieving a *win-win* collaboration between the app and its users.

**Self-driving Cars** As highlighted by self-driving car companies (e.g., Waymo [1,48] and Zoox [14]), handling unprotected left turns is a challenging task for this rapidly emerging technology since it requires not only human-level driving skills but also even psychological understanding [47]. For example, some pedestrians may let left-turning cars go first, while others may ignore or fail to see left-turning cars and cross the street without paying attention to the approaching cars and pedestrians. In such a case, both human drivers and pedestrians could use a body language or even eye contact to resolve this conflict, but it is very challenging for autonomous cars to understand and resolve this type of conflict.

Hence, autonomous cars can avoid such intersections by using `TurnsMap` to enhance safety for both cars and pedestrians by selecting paths with a minimum number of unprotected left turns in the path planning process [30].

### 4.2.2 Why Mobile Sensing for Classifying Protected/Unprotected Left Turns?

`TurnsMap` outsources the data collection task to the crowd and only processes the IMU sensor data, thus can enable a much larger coverage and faster update-rate than the existing road-survey (e.g., mapping cars of Google Street View) and camera-based approaches, which collect road images by using cameras on either dedicated mapping car or users' smartphones. Specifically, the performance of understanding traffic scenes (i.e., classifying left-turn enforcements) from image data — one of the most common data sources for legacy scene perception tasks, could be hampered by both diverse real-world settings and limited resources:

- *Poor visibility of traffic signals/signs:* The visibility of traffic signals/signs may be distorted due to the weather, lighting condition and even local regulations. Moreover, the location of traffic lights/signs may be different in different regions — they could be located at the middle of an intersection or above a waiting line (e.g., in European countries). In fact, determining a region-of-interest (ROI) based on computer vision is still a challenging open problem [131]. The complexity of accounting for real-world environments makes it even harder for a computer vision task to function correctly on a large and diverse image dataset.
- *Limited coverage:* Due to the different level of development, a vast majority of regions in the world cannot be covered by dedicated road survey services. For example, as of 2017, only a small number of areas are fully covered by Google StreetView [9]. In fact, many countries and regions in Asia and Africa are not yet covered, nor planned to be covered in the near future.
- *High cost/overhead of collecting road image from smartphones:* Processing image data requires high computational power and is thus expensive to run locally on the user's smartphone. Even if the developer chooses to upload the image(s) to the cloud service for processing, transmission of the bulky image and/or video data may exhaust mobile network bandwidth. Moreover, to have a clear view of an intersection, the user has to fix the phone on the windshield, which is a stringent and often undesirable requirement for various use-cases.

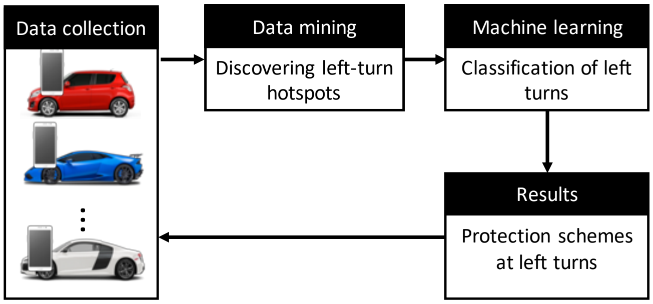

Data collection
Data mining
Discovering left-turn hotspots
Machine learning
Classification of left turns
Results
Protection schemes at left turns

Figure 4.2: An overview of TurnsMap

The design of TurnsMap aims to collect and publicize the safety-critical left-turns information at low overhead/cost and eventually enhance traffic safety at scale. TurnsMap will be detailed in Secs. 4.5 and 4.6.

## 4.3 Overview of TurnsMap

As shown in Fig. 4.2, TurnsMap is comprised of three key building blocks: the *data collection module* for collecting mobile data (i.e., IMU + GPS) from users, *data mining module* for constructing the database of left turns, and *machine learning pipeline* for differentiating left-turn protection schemes. First, TurnsMap collects IMU sensor and GPS data from the mobile devices carried by drivers and passengers (Sec. 4.4). Next, the data mining module (Sec. 4.5) extracts left turns from these data and identifies the road segments, called *left-turn hotspots*, with high-density left-turn driving data traces. To classify protected/unprotected left turns based on people's driving behavior, we devise a machine learning pipeline. For fast and accurate collection of ground truths for the machine learning task, we take a human-in-the-loop approach by outsourcing annotation (i.e., ground truth labeling) tasks on Amazon Mechanical Turk. These efforts together help construct a dataset for traffic scene understanding at intersections. Finally, we propose a recurrent neural network-based model to learn the classifier for differentiating left-turn protection schemes. Note that TurnsMap can help enrich navigation app's functionality (Sec. 4.8) and enhance users' driving safety. Allowing the TurnsMap framework to intrinsically incentivizes users of automotive apps (e.g., navigation apps on smartphones) to participate in data collection for building a continuously-expanding database. This feature helps formulate a win-win collaboration between the user and `TurnsMap`.

## 4.4 Collection of Driving Data

`TurnsMap` classifies left turns using smartphone IMU sensor and GPS data (i.e., the longitude and latitude pair). Specifically, `TurnsMap` needs a time series of sensor data collected from the user's phone while driving. A ground truth dataset of left-turn protection settings is also needed for building and testing the machine learning algorithm. Since no datasets are available for developing and testing `TurnsMap`, we collected and constructed a dataset. Our driving data collection methodology is introduced in this section and our approach for collecting the ground truth is highlighted in Sec. 4.6.

**Collection of Real-world Driving Data from Smartphones** There are three key rationales behind the design of our mobile app for collecting the natural driving data. First, different phone postures (e.g., sitting on a phone mount or cupholder) can affect the IMU sensor readings. For example, if the smartphone's screen is facing down, its yaw axis angular speed will be the inverse of the readings with the screen facing up. Second, since smartphones have limited resources (e.g., CPU, battery, and cellular network bandwidth), the data collection should incur low overhead on the users' devices — a high overhead will likely discourage the users in adopting and/or contributing to `TurnsMap`. Third, a comprehensive database should cover heterogeneous intersection setups, i.e., the dataset should cover different drivers since different driving habits may incur diverse driving maneuvers at left turns. This is an essential requirement for building a comprehensive classifier that is resilient to the changing environment and users.

To handle different phone postures inside a car and achieve consistency in data analytics, Our data collection app aligns the phone coordinate with the earth coordinate

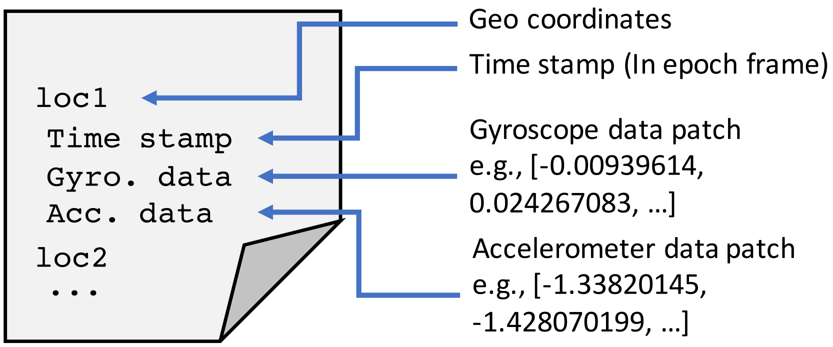


Figure 4.3: The snapshot of the data the app collected.

by using the coordinate transformation as described in [102]. This capability enhances TurnsMap's usability by eliminating the restriction of the device's posture, i.e., a phone can be placed at any stable position, such as on a cupholder/phone mount, inside the car.

Lowering the overhead on the user's smartphone is another key objective of our data collection app. Specifically, the app samples the thus-aligned IMU sensor data and GPS (i.e., (longitude, latitude) pair) data at 100Hz and 1Hz,[2] respectively. The snapshot in Fig. 4.3 shows the data format — one geolocation sample includes a timestamp vector, a gyroscope data vector, and an accelerometer data vector, where each vector consists of 100 data samples. Because of the light-weight feature of IMU and GPS data, our collection process incurs low overhead on users' mobile devices; our on-road data collection is shown to generate only 6.2MB data per hour. To enhance the flexibility of using their data plan, the participants can choose to upload the data in real time, or when a free Wi-Fi connection is available. Finally, the data collection app is shown to incur only marginal CPU and energy overheads (Sec. 4.7.2).

We also aim to enrich the diversity (e.g., different driving behaviors and car models) of our dataset from the participants' perspective. To this end, we recruited 18 drivers (14 males and 4 females of age ranging from 23 to 70) and installed our data collection app on their smartphones. The participants are students, university professors, and company employees.

[2] 1Hz is the default sampling rate on the current location chipset embedded in smartphones.

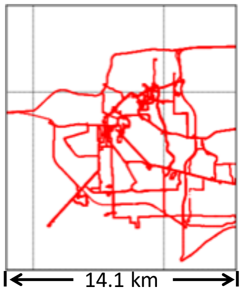


Figure 4.4: Suburban driving trace.

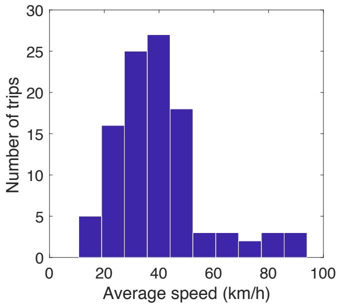


Figure 4.5: Histogram of the averaged driving velocity.

We used 10 different car models, including coupe, sedan, SUV, and sporty cars. To collect natural driving data, no restriction was imposed on driving route, vehicle type, smartphone model and location, etc. That is, the participants were told to drive their own cars for daily commute as normally as possible. To secure the participants' privacy and data safety, we applied and received our university's IRB approval (registration number: IRB00100245).

The data collection app has been collected data in a region that includes both urban and suburban environments in Ann Arbor and Detroit metropolitan area in the U.S. Our data-collection process has thus far yielded 1.6GB mobile IMU data, covering an area of approximately 300km$^2$, with the accumulated travel distance of more than 3,589 km. The data collection was conducted between 7:30 am to 6:30 pm. Each driver Here we show a suburban area of driving data we collected in Fig. 4.4. The red traces are the driving trajectories. The dataset includes 105 trips contributed by all participants, with the accumulated driving time 78.3 hours. The histogram of the average driving velocity is shown in Fig. 4.5. The majority of the trips has average speed between 20 – 45 km/h, reflect the normal average driving speed in cities. There are a few trips have high averaged speeds, e.g., exceed 80 km/h. These trips were collected from driving on highways.

## 4.5 Mining the Driving Data

TurnsMap infers left-turn protection settings by using the mobile sensor data. We need to answer the following questions for this inferencing. How to differentiate intersection settings by using the mobile sensory data and how to organize the crowdsourced data for this purpose? To answer the first question, we elaborate how IMU sensor readings can capture the interruptions between left-turning cars and the crossing traffic and/or pedestrians. These interruptions are then used to differentiate left turn setups. To answer the second question, we use data mining to extract and cluster the left-turn data snippets to discover left-turn *hotspots*, road segments with many left turns. Note that a hotspot could be an intersection and even an entrance of a popular plaza. These answers are essential for constructing a comprehensive dataset of left turns for the subsequent machine learning.

### 4.5.1 Differentiating Left-Turn Enforcements from IMU Sensor Readings

The key for distinguishing protected and unprotected left turns is the fact that left turns are likely to be interrupted on road segments without strict left-turn enforcements, such as left-turn signals or stop signs. Based on this observation, we uncover the root causes of interruptions which are identified by utilizing phone IMU sensor data.

The root causes of these interruptions are (i) the enforcement-free intersection setup to increase the traffic throughput[3] and (ii) pedestrians crossing the street the car is left-turning to. Due to the low priority given to left-turning cars, their drivers need to pay full attention to any sudden situation change and prepare to pause and/or yield to the oncoming traffic and pedestrians crossing the street they are turning onto.

Fig. 4.6 (a) shows possible interruptions that a car may experience while turning left at an intersection with a unprotected enforcement (i.e., circular traffic light in Fig. 4.6 (a)). The most common (as stated in the driver manual [44]) left-turn maneuver works as follows. A left-turning car first enters the intersection after the green light comes on and then suspends its turn if there is an oncoming car from the opposite direction. Upon resuming the left turn, the car needs to pay attention to, and prepare to pause for, pedestrians

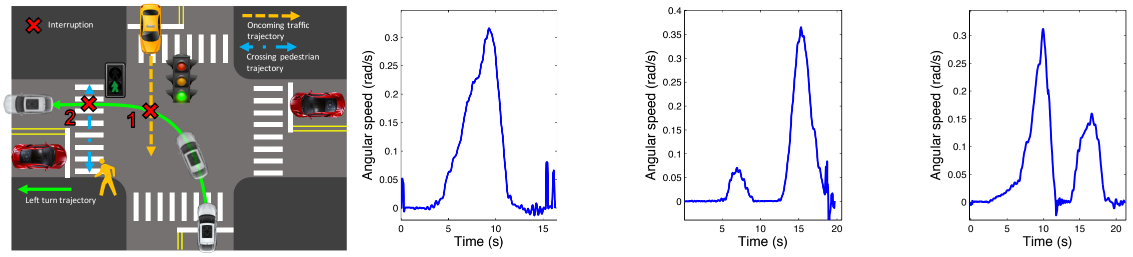
[3] Oncoming cars from the opposite direction share the same green light phase with left-turning cars.

crossing the street/road it is turning onto. Finally, the left-turning car completes the turn by exiting the intersection.

Similar interruptions also occur at the intersections with stop signs. After making a full stop at the sign, the left-turning car needs to proceed until the driver can see the approaching traffic. In summary, interruptions are likely to happen while making unprotected left turns at intersections.

In contrast, interruptions occur rarely at intersections with protected enforcements, because the protected traffic signal (i.e., the left-turn traffic arrow) grants left-turning cars an exclusive traffic phase, hence preventing pedestrians and oncoming vehicles from crossing the intersection. In fact, as stated in the NHTSA intersection engineering handbook [30], one of the main purposes of enforcing left-turns protection is to reduce the left-turning cars' conflicts with the oncoming traffic and crossing pedestrians.

`TurnsMap` uses the smartphone's gyroscope to detect the interruptions. Since an interruption is essentially a pause of steering maneuver, the coordinate-aligned gyroscope data (i.e., angular speed) can capture the differences between normal and interrupted left-turns. Fig. 4.6 (b)-(d) shows the gyroscope readings of three scenarios: no interruption, interruption at the middle of an intersection (interruption 1 in Fig. 4.6 (a)), and a left turn with an interruption at the crosswalk (interruption 2 in Fig. 4.6 (a)). When a car makes a left turn, the gyroscope reading captures the car's angular speed ($\omega$). As depicted in Fig. 4.6(b), when the driver makes a (normal) left turn without interruption, s/he first turns the steering wheel counter-clockwise, creating a monotonic increase of the vehicle's angular speed $\omega$. The driver then makes a clockwise turn of steering wheel until the car's orientation returns to straight, or $\omega$ returns to approximately 0. Hence, a smooth left turn generates a single *bump* in the gyroscope reading. When an interruption occurs, the rise of $\omega$ stops and returns to 0, thus creating the first bump in the gyroscope reading; after the driver resumes the left turn, $\omega$ will first rise and then drop until the driver exits the intersection, creating the second bump in the gyroscope reading. Figs. 4.6(c) and (d) show



(a) Left turn interruptions(b) No interruption (c) Interruption 1 (d) Interruption 2

Figure 4.6: (a) Possible conflicts at an unprotected left-turn. The gyroscope readings shows (b) no interruption, (c) an interruption occurs at the middle of intersection, and (d) an interruption occurs at a crosswalk, respectively.

this unique pattern of the gyroscope reading.

The above finding/observation is the cornerstone of `TurnsMap`, showing that IMU data collected during each left-turn contains information useful to capture the left-turn enforcement information.

### 4.5.2 Extraction of Left Turns from Mobile Sensing Data

The size of the crowdsourced time-series data can be massive, and some portions of the data may be redundant and not useful for `TurnsMap`, i.e., driving on straight roads. Hence, it is necessary to extract data snippets that contain the targeted driving maneuver (i.e., left turn).

To achieve this goal, `TurnsMap` extracts left turns from the IMU and GPS sensor data of the driver's smartphone. Specifically, we propose a novel left-turn detection scheme based on the V-Sense algorithm [102] as detailed in Chapter II.

Let us first review how the V-Sense algorithm detects left turns from the mobile sensing data. V-Sense detects vehicle steering maneuvers by harvesting the morphological patterns of the sequential[4] gyroscope data. First, V-Sense analyzes the gyroscope data to detect the start and end points of a "bump"-shaped curve by using a predetermined threshold derived from the common driving behavior. Next, to validate if the current bump is incurred by a driving maneuver or noise (e.g., system glitch), V-Sense examines the statistical features

[4] In this chapter, we use the terms "sequential data" and " time-series data" interchangably.

(e.g., duration, magnitude) of the bump. V-Sense also handles false detections. Specifically, since some vehicle movements (e.g., driving on a curvy road, or up/down hill) may also regard a bump-shaped curve as a turning maneuver. To detect these false detections, V-Sense calculates the moving vehicle's displacement and uses it to validate the detection result. By performing the above steps, a turning maneuver can be detected with *linear* time complexity. We refer the interested readers to Sec.3.2 of [102]. Compared to other sequential data analytics schemes, this method has the following advantages:

- **Resilience to the changing environment:** although approaches based on location information (e.g., GPS data) can detect the vehicle's heading and thus inferring turns, their performance depends on the received satellite signal strength (as shown in [98, 117]). So, they could result in varying performance due to the changing real-world environment.
- **Linear time complexity:** V-Sense [102] uses the statistical feature in time-series gyroscope data to detect the "bump" (as shown in Fig. 4.6) that reflects turns, thus only incurring linear time complexity. Compared to legacy time-series classification algorithms such as *k*-nearest-neighbors dynamic-time-warping (knn-DTW [29]), which has quadratic time complexity, V-Sense is more efficient in processing large-scale crowdsourced time-series data;
- **Easy parameter setting:** Unlike existing time-series data classifications, V-Sense does not require any pre-defined pattern but uses the thresholds derived from a natural driving model.

However, `TurnsMap` faces a unique challenge — one bump may not contain a complete left turn since the turn can be interrupted (e.g., by crossing pedestrian/cars), thus generating two or more bumps in a left turn. To overcome this challenge and accurately crop the sensor data trace that contains a complete left turn, we examine whether or not the neighboring bumps are geo-located adjacently. Specifically, if neighboring bumps on the same IMU trace are adjacent in geo-distance, i.e., great-circle distance between two points on the surface of the earth, then they together represent a single left-turn maneuver. In this line, we propose a two-stage bump detection process. Upon detecting a bump by the V-Sense algorithm, whether the bump represents a complete left turn is not yet determined. To determine if the pending bump's geolocation is close to the neighboring bump(s), we check if the geo-distance between the centroids of geolocations of those bumps (let $loc_{cur}$ and $loc_{pend}$ denote the centroid of the geolocation of the current bump and the pending bump, respectively) is within a threshold $\theta_{geo}$.[5] The centroid is derived by using $FindCentroid(\cdot)$, which is the mean of latitude and longitude readings. Note that this averaging step can also mitigate the fluctuation of GPS readings. The geo-distance is calculated by using the haversine formula [19]. Thus, if the current and pending bumps are from the same left turn, we merge the current bump with the pending bump(s). Otherwise, the current bump is a complete left turn. Finally, the start and end timestamps (i.e., $t_{cur,start}$, and $t_{cur,end}$) of the current bump can be used for cropping the left-turn data snippets from all necessary time-series data traces — gyroscope, accelerometer, and geolocation data. For example, a framed The left-turn detection algorithm is summarized in Algo. 2.

We can form the output of left-turn detection as follows. To organize the thus-extracted heterogeneous data (e.g., IMU sensor and geolocation traces), for *each* left-turn maneuver, we store its data as a tuple called *left-turn tuple* $\boldsymbol{l}$:

$$\begin{aligned} \boldsymbol{l}_k &= \{c_k, \boldsymbol{gyro}_k, \boldsymbol{acc}_k, \boldsymbol{loc}_k\}, \\ \boldsymbol{L} &= \{\boldsymbol{l}_1, \cdots, \boldsymbol{l}_k, \cdots, \boldsymbol{l}_K\} \end{aligned} \tag{4.1}$$

where $k \in \{1, \ldots, K\}$, $K$ is the total number of left turns extracted by Algo. 2; $c_k$ is the centroid of the $k$-th left turn; $\boldsymbol{gyro}$ and $\boldsymbol{acc}$ are IMU sensor vectors cropped from IMU sensor traces; $\boldsymbol{loc}$ is the geo-location data trace. $\boldsymbol{L}$ is the list that stores all left-turn tuples. This extraction process discovered $K = 68,266$ left-turn maneuvers from the raw collected IMU data.

[5] $theta_{geo}$ is derived from the diameter of road intersections. We use $\theta_{geo} = 30$m.

**Algorithm 2** Left-turn detection algorithm of `TurnsMap`

```
Input gyroscope ($\Omega$), accelerometer ($\Phi$), and location ($\Gamma$) data traces
for $\omega$ in $\Omega$ do
    Detect a valid bump based on V-Sense algorithm
    Find the timestamp range of the current bump $[t_{cur,start}, t_{cur,end}]$
    $loc_{cur} = FindCentroid(\Gamma_{t_{cur,start}}^{t_{cur,end}})$. Here, $\Gamma_{t_{cur,start}}^{t_{cur,end}}$ is the cropped location data trace based on timestamp
    if $Haversine(loc_{cur}, loc_{pend}) \leq \theta_{geo}$ then
        /*Update the bump information*/
        $t_{cur,start} = t_{pend,start}$
        $loc_{cur} = FindCentroid(\Gamma_{t_{cur,start}}^{t_{cur,end}})$
    else
        /*$Range[t_{cur,start}, t_{cur,end}]$ includes a complete left turn*/
        Extract IMU and location data snippets (i.e., $\boldsymbol{gyro}$, $\boldsymbol{acc}$, and $\boldsymbol{loc}$) respectively.
        Calculate the centroid of this left turn: $c_k = FindCentroid(\boldsymbol{loc})$
    end if
    Update the timestamp range: $t_{pend,start} = t_{cur,start}$, $t_{pend,end} = t_{cur,end}$, $loc_{pend} = loc_{cur}$
end for
```

### 4.5.3 Construction of Left-turn Hotspots

It is not possible to capture the left-turn setting of an intersection based only on discrete left-turn traces. There could also be false detections of left turns in the left-turn extraction step; for example, a swerving maneuver at a parking lot could be classified as a left-turn maneuver. To address this problem, we cluster multiple left-turn traces to form *left-turn hotspots* — road segments with left turn maneuvers.

We propose a DBSCAN-based clustering [114] method to discover left-turn hotspots from $\boldsymbol{L}$. DBSCAN is a density-based algorithm that does not require a pre-defined number of clusters and is proven to perform well in clustering spatial data. In our case, $DBSCAN(\cdot)$ uses the centroid of each left-turn instance as the clustering criterion. The output (i.e., $\boldsymbol{C}$) of $DBSCAN(\cdot)$ is a collection of clusters, which is comprised of left-turn tuples ($\boldsymbol{l}$s). We only keep those clusters with “enough” left turns — any cluster with less than 5 left-turn tuples are discarded owing to the insufficient number of traces. Finally, for each cluster, we group its gyroscope, accelerometer and location traces, respectively, to derive left-turn hotspots. Our clustering algorithm is summarized in Algo. 3.

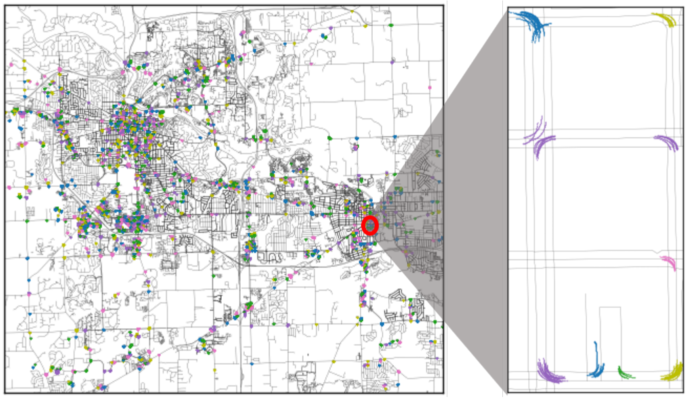


Figure 4.7: Clustering result of DBSCAN. Each cluster of left-turn traces is a left-turn hotspot.

Each of the thus-constructed left-turn hotspot can be represented by a data tuple $\boldsymbol{x}$:

$$\begin{aligned} \boldsymbol{x}_i &= \{c_i, \boldsymbol{Gyro}_i, \boldsymbol{Acc}_i, \boldsymbol{Loc}_i\}, \\ \boldsymbol{X} &= \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_N\} \end{aligned} \tag{4.2}$$

where $i \in \{1, \ldots, N\}$, $N$ is the number of left-turn clusters with more than 5 traces. $\boldsymbol{Gyro}$, $\boldsymbol{Acc}$, $\boldsymbol{Loc}$ denote the thus-aggregated gyroscope, accelerometer, and location traces, respectively. $\boldsymbol{X}$ is the list of left-turn hotspot tuples.

**Algorithm 3** Cluster Left Turn Traces

```
Input:
$\boldsymbol{C}$ = $DBSCAN(\boldsymbol{L})$
for each cluster in $\boldsymbol{C}$ do
    if Number of left-turn instance in this cluster is less than 5 then
        /*Discard this cluster*/
    else
        $c$ =$FindCentroid$(centroids of left-turn tuples in this cluster)
        Aggregate $\boldsymbol{gyro}$, $\boldsymbol{acc}$, and $\boldsymbol{loc}$ of each left-turn tuple into set $\boldsymbol{Gyro}$, $\boldsymbol{Acc}$, and $\boldsymbol{Loc}$, respectively.
        The data tuple of this left-turn hotspot is $\{c$, $\boldsymbol{Gyro}$, $\boldsymbol{Acc}$, $\boldsymbol{Loc}\}$.
    end if
end for
```

Fig. 4.7 shows part of our clustering result. The location traces are interpolated to show the trajectory of a left-turning car. In the zoom-in view, one intersection is not covered due to lack of driving data, but the other intersections are clustered by DBSCAN. We evaluate the performance of our hotspot extraction in Sec. 4.6.1.

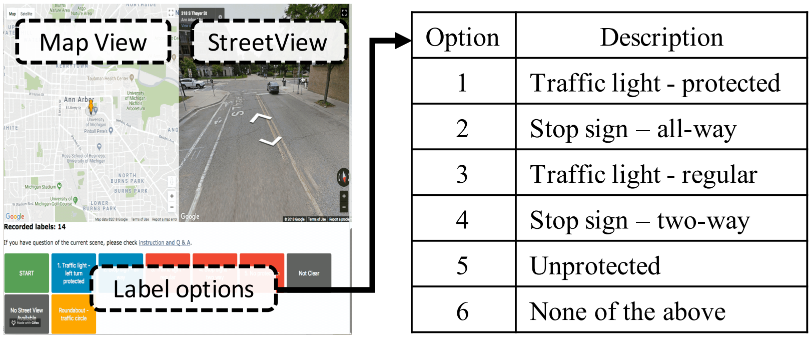
## 4.6 Deep Learning Framework

Thus far, we have constructed a dataset $\boldsymbol{X}$ of time-series traces at left-turn hotspots. To enable classification of left-turn settings by using the time-series data, we first collect the ground truth of each left-turn hotspot in $\boldsymbol{X}$, and then use a deep learning pipeline to train the classifier.

### 4.6.1 Collection of the Ground Truth

Due to the lack of the ground truth of each hotspot's left-turn setting, we need to collect the ground truth from sratch. A human-in-the-loop method is used to collect this information efficiently. Specifically, we outsource annotation tasks to the crowd on Amazon Mechanical Turk (AMT) and use Google StreetView — a large-scale image database that includes traffic scene information collected by mapping cars — as the accessible reference for the annotators to identify the left-turn protection setting of each hotspot.

An easy-to-follow annotation process is essential to minimize the participants' confusion for good quality of annotation. To meet this requirement, we design an interactive labeling/annotation webpage as the participants' working platform. Described below is how to annotate each left-turn hotspot on this webpage. Before starting the annotation, each AMT participant will be asked to read our instruction (as shown in Fig. .1 in Appendix) carefully to understand the annotation process. Next, our backend system (a php script running on the server for hosting the webpage) will randomly pick a hotspot $\boldsymbol{x}$ from $\boldsymbol{X}$ and then display the corresponding Google StreetView based on the centroid of $\boldsymbol{x}$. The annotator then needs to inspect the StreetView (i.e., by zooming in/out and changing the viewing angle) to get a clear view of the left-turn protection setting. The annotator will



Recorded labels: 14

If you have question of the current scene, please check instruction and Q & A.

| Option | Description |
|---|---|
| 1 | Traffic light - protected |
| 2 | Stop sign – all-way |
| 3 | Traffic light - regular |
| 4 | Stop sign – two-way |
| 5 | Unprotected |
| 6 | None of the above |

Figure 4.8: Online annotating system of `TurnsMap`. The left figure is a screenshot of its interface (the instruction section is omitted due to space limit). The right table shows the options available to the annotators.

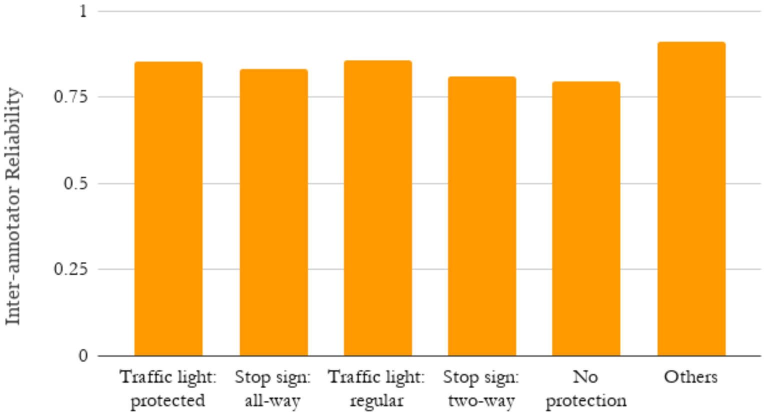


Figure 4.9: Inter-annotator reliability of each type of left-turn protection.

then be asked to choose one of the six options in the table of Fig. 4.8. Option 1 represents the protected left-turn signal; options 2–5 the unprotected left turns; option 6 none of the five aforementioned options according to the participant's perception/judgement, e.g., parking lots and/or road segments without any StreetView image. Finally, our backend system will record the annotator's input, and a new hotspot will be displayed on the webpage.

After publishing the annotation task on AMT, we recruited 231 annotators, and asked each of them to *independently* annotate/label 30 randomly-selected hotspots from $\boldsymbol{X}$. Each annotator spent an average of 26.5min to complete the task, which is reasonable for a new annotator to understand and annotate hotspots.

Although crowdsourcing the collection of ground truths via AMT can significantly speed up the annotation, erroneous annotations are inevitable since the AMT participants are not domain experts and may have different levels of perception. Thus, we try to enhance the quality of the collected annotations as follows.

- Recruiting proficient annotators. We ensure the annotators' competence by only recruiting "master" workers [2] — elite workers who have demonstrated superior performance in their previous tasks and have been acknowledged by the corresponding requesters. We must also pay the master workers more to incentivize their participation.
- Ensuring participants' clear understanding of the annotation task. A thorough and clear understanding of the annotation task can improve the participant's performance. To meet this goal, we only recruited drivers who reside in the U.S., since our driving data is collected from a US metropolitan city. Our easy-to-follow instruction helps the annotators understand different left-turn protection mechanisms and the annotation process.
- Aggregating the collected annotations via majority voting. Annotated data need to be filtered and refined for accuracy. We first ensure each left-turn hotspot received multiple annotations from the participants. Specifically, we discard any left-turn hotspot with less than 3 annotations. Then, for each remaining hotspot, we apply simple majority voting to determine its annotation. That is, we select the annotation with the largest number of repetitions. Note that a similar criterion is also commonly used in other large-scale ground truth collections, e.g., ImageNet [109] and DeepDrive [179].

We randomly select 1,000 hotspots from $\boldsymbol{X}$ and collected 6,016 annotations — each hotspot has an average of 5.47 annotations. We assess the quality [152] of our dataset by inspecting the inter-annotator reliability of each left-turn protection setting, which measures the consistency among different annotators. It is a commonly used evaluation

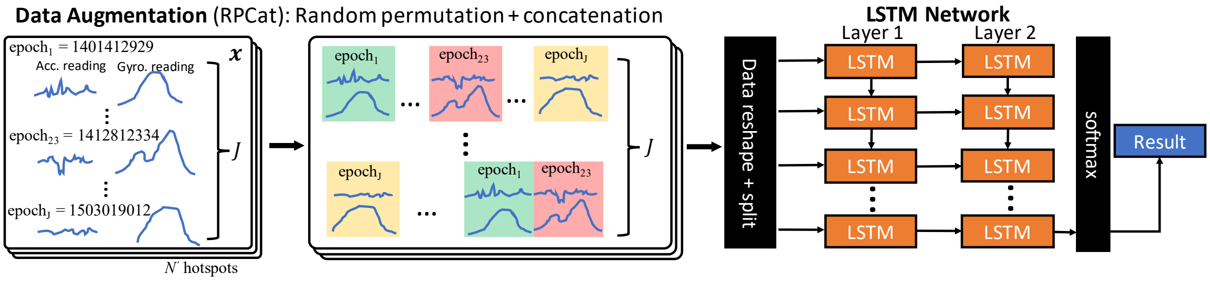


Figure 4.10: The machine learning pipeline. The left frame shows the gyroscope and accelerometer traces in a left-turn hotspot $\boldsymbol{x}$; the middle frame shows RPCat permutation and concatenation of IMU data in $\boldsymbol{x}$; the right-hand side network shows the LSTM-based network architecture.

metric for assessing the quality of annotations collected from the crowd [128]. As shown in Fig. 4.9, the inter-annotator reliability of each category is around 85%.

Moreover, the annotation result can also help us understand the performance of our left-turn hotspot extraction (as stated in Sec. 4.5). Since false detections of a hotspot may be due to cars' swerving in open areas (e.g., parking lots), we can use the annotators' feedback on each hotspot to inspect if the hotspot is indeed a road segment that can facilitate left turns. According to the aggregated annotation results, 96.5% (965 out of 1,000) of the annotated hotspots are road segments that can be categorized into one of the five scenarios as shown in Fig. 4.1. Our left-turn hotspot extractions shown to be able to accurately identify road segments that can facilitate left turns.

Finally, we construct our dataset by using the annotated hotspots. Specifically, for hotspot $i$, we denote its ground truth as $y_i$ if it has an annotation that passed our quality control test. Thus, we can have

$$\begin{aligned} \boldsymbol{X} &= \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_{N'}\}, \\ \boldsymbol{Y} &= \{y_1, \ldots, y_i, \ldots, y_{N'}\} \end{aligned} \tag{4.3}$$

where $y \in \{1,2,3,4,5\}$ as shown in Fig. 4.8; $i \in \{1,\ldots,N'\}$, $N'$ is the number of valid hotspots with valid annotations. In the current dataset, $N' = 965$. Note that by associating

$\boldsymbol{X}$ with $\boldsymbol{Y}$, we can have $(\boldsymbol{x}, y)$ pairs, which can be applied in supervised machine learning.

### 4.6.2 Formulation of the Learning Problem

Classification of protected/unprotected left turns based on mobile sensing data at hotspots ($\boldsymbol{X}$) can be cast as a supervised learning problem for binary classification of time-series data. We modify $\boldsymbol{Y}$ to form a binary classification task: label 1 (as shown in Fig. 4.8) is categorized as a protected left turn while labels 2–5 are categorized as unprotected left turns. That is, $y_i =$ {Protected, Unprotected}, $i \in \{1, \ldots, N'\}$, $N' = 965$. As a result, we have 206 protected and 759 unprotected hotspots. The supervised machine learning problem can be formally state as follows.

**Problem statement.** Given $N'$ pairs of data of the form $\{(\boldsymbol{x}_1,\ y_1),\ \ldots,\ (\boldsymbol{x}_{N'},\ y_{N'})\}$, the objective is to learn a model $\boldsymbol{X} \rightarrow \boldsymbol{Y}$, where $y$ is an annotation, $\boldsymbol{x}$ is the measurement represented as a tuple of several time-series traces.

The key challenge in this learning task is how to form the input data by using the observation tuples (shown in Eq. (4.2)). The input data formation is different from many existing time-series classification tasks, such as pattern matching [97], and DeepSense [176] — a recent light-weighted deep learning framework which is optimized for running on mobile devices. Specifically, in the legacy time-series classification, each observation $\boldsymbol{x}$ is a single data trace (vector). For example, to train a classifier for the GaitID task (identifying a user based on his/her gait traces), each observation in the training set is a 2-dimensional vector of IMU sensor readings. However, in `TurnsMap`, each observation is a *tuple* of data traces. As shown in Eq. (4.2), the data of each observation $\boldsymbol{x}$ is comprised of $\boldsymbol{gyro}$ and $\boldsymbol{acc}$, which are sets of gyroscope and accelerometer data vectors of different lengths. Without such data formation, it is not possible to distinguish left-turn settings based on one time-series data vector. For example, the left side frame in Fig. 4.10 illustrats the accelerometer and gyroscope data traces collected from an unprotected left turn. Note that not all traces exhibit the interruption pattern, i.e., some of the collected time-series traces are as smooth as if they were collected from a protected left turn. Therefore, a machine learning pipeline needs to pre-process the time-series data tuple to form data that can be used as input for the machine learning algorithm.

### 4.6.3 Deep Learning Pipeline

To address these challenges, we propose a deep learning pipeline that is comprised by a novel data augmentation process and a deep Recurrent Neural Network (RNN) based on Long-Short Term Memory (LSTM) cells.

#### 4.6.3.1 Data Augmentation.

The goal of the data augmentation [126] is to construct and expand the training dataset without undermining the underlying pattern. For this purpose, we design *randomly permuted concatenation* (RPCat) for each hotspot data, i.e., $\boldsymbol{X}$. As shown in the second frame on the left of Fig. 4.10, RPCat has two steps: (i) expand the set of observations by permuting the sequence within $\boldsymbol{x}$; (ii) concatenate the data traces of each permuted observation. This approach is inspired by [139] in the area of natural language processing (NLP), where data snippets (e.g., phrases in NLP) are concatenated together to form an overview of the overall dataset.

Let us illustrate RPCat using an example shown in Fig. 4.10. Suppose a hotspot's data $\boldsymbol{x}$ is comprised of $J$ gyroscope and $J$ accelerometer traces (i.e., $J$ is the number of the clustered left-turn maneuvers at this hotspot). The sequence of data traces in $\boldsymbol{x}$ is originally organized according to the timestamp (we use epoch time in `TurnsMap`) of the collected data. Note that different left turns at this hotspot are independent of each other. That is,

$$P(T_1,\ldots,T_j,\ldots,T_J) = P(T_1)\cdots P(T_j)\cdots P(T_J), \tag{4.4}$$

where $T_j$ is the $j^{th}$ left-turn event and $P(T_j)$ is the probability of $T_j$'s occurrence. Due to the independence of different left turns, the arrival sequence of these data traces should *not* affect the classification result.

RPCat reflects the this insight and reconstructs $\boldsymbol{x}$ as shown in the second frame in Fig. 4.10. Specifically, RPCat firstly *permutes* the sequence of data traces in $\boldsymbol{x}$ and then *concatenates* the data. After each permutation, RPCat generates a new sequence of data traces. Finally, to unify the data size for machine learning, we resize each trace to length $FixLen$. We repeat RPCat $J$ times to obtain a $J \times FixLen \times 2$ tensor for $\boldsymbol{x}$. We execute the above process for all $N'$ hotspots. Finally, after reshaping and splitting the data based on batch size, we transfer the data into the input tensor for the LSTM network.

The parameters of the current RPCat are set as follows:

- **Repetition of permutation.** This parameter is to balance the dataset. For example, unprotected hotspots constitute the majority of the current dataset due to their prevalence in the real world. To avoid an imbalanced dataset, if $\boldsymbol{x}$ is an unprotected hotspot, we repeat the permutation $J$ times, whereas for each protected hotspot, we permute $2J$ times. This allows us to have a more balanced dataset, which is necessary for better performance in training the LSTM network. We do not repeat RPCat full permutation times because permutation of $J$ unique elements ($Perm(J,J)$) can be very large, e.g., $Perm(10,10) = 3,628,800$. Such a large dataset would be imbalanced and can take prohibitive long time for training.
- **Length of data traces.** Our current implementation uses $FixLen = 800$. The optimization of the unified data length is left as our future work.

It is importation to note that the permutation does *not* undermine the sequential pattern of the data for `TurnsMap`. As highlighted in Sec. 4.5, the sequential pattern (e.g., interruptions) of left-turn data resides in each left-turn maneuver. Since the permutation only shuffles the order of left turns, it does not change the sequential pattern of each left-turn data.

Now, the remaining question is how to learn the pattern based on the augmented data.

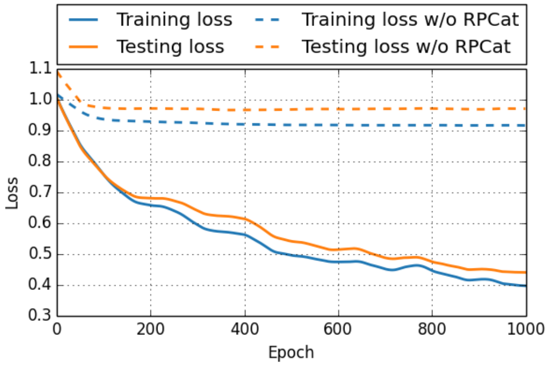


Figure 4.11: Efficacy of RPCat on training with LSTM.

#### 4.6.3.2 RNN Framework.

We use a deep RNN framework with LSTM cells to learn the underlying pattern(s) from the augmented data. As a specific type of RNN architecture, LSTM has proven advantages [123] over other machine learning algorithms, such as SVM, random forest, convolutional neural network, etc., for classifying sequential data. Specifically, each LSTM cell uses a forget gate to adjust the extent of passing the state variable to the next unit — an important feature for capturing dependencies through time [132]. Moreover, the forget gate design is shown to have a desirable effect on mitigating the vanishing gradient problem [121], a common limitation for the classical RNN architecture that may restrict the network from learning the pattern from sequential observations. We used stacked LSTM since a deeper hierarchical structure enables the network to gain more accurate understanding of the intrinsic pattern by disseminating the learning task to each layer [123, 129].

The hyper-parameter settings of our network are: 2 LSTM layers, 32 LSTM cells per layer, batch size of 300, and the training rate of 0.001. We use the tanh function for updating the cell and hidden state, whereas the sigmoid function is used for the gates in LSTM cells. To mitigate overfitting, we applied a 0.5 dropout rate in each LSTM layer. The Adam optimizer [140] is used for iteratively updating network weights. Finally, the network feeds the thus-learnt feature vector ($1 \times 32$) into a softmax layer for generating the classification result.

Now, we first test the efficacy of our RPCat + deep LSTM pipeline by evaluating the training loss. To obtain the test data, we randomly selected 20% of hotspots from $\boldsymbol{X}$ and then applied RPCat to build an augmented testing set. We have implemented our learning pipeline with TensorFlow on our lab server equipped with one Titan Xp GPU. Fig. 4.11 shows the efficacy of RPCat and also the training and testing loss (training/testing loss is the penalty of misclassification of training/testing set [121]) by using our learning pipeline. Without applying RPCat, both training and testing losses converged to high values, indicating underfitting — the LSTM network does not have enough data for learning the underlying pattern. With RPCat, despite the fluctuation caused by the convergence feature of Adam optimizer, both training and testing losses decrease with the increasing number of epoch (in neural network epoch, one epoch means the number of times the entire dataset is processed by the algorithm). Specifically, training and testing losses are stabilized at 0.4 and 0.45, respectively, after about 900 epochs. We will evaluate the performance of our machine learning pipeline in Sec. 5.4.

## 4.7 Evaluation

We first evaluate the overhead of `TurnsMap`, then the performance of our deep learning pipeline, and finally present the insights gained from the evaluation.

### 4.7.1 Performance Metrics of TurnsMap

#### 4.7.1.1 Evaluation Metrics of the Deep Learning Pipeline.

We evaluate our machine learning pipeline's performance by using the cross-validated performance metrics. Fig. 4.12 shows the confusion matrix where each row represents the actual class; each column represents the predicted class. Fig. 4.13 shows the precision, recall and F-1 score, where precision represents the ratio of true positive detections to the

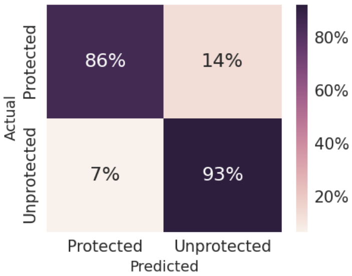


Figure 4.12: The normalized onfusion matrix for analyzing `TurnsMap`'s performance.

| Category | Precision | Recall | F-1 |
|---|---|---|---|
| Protected | 0.90 | 0.86 | 0.88 |
| Unprotected | 0.91 | 0.93 | 0.92 |
| Average | 0.90 | 0.90 | 0.90 |

Figure 4.13: Performance metrics of the LSTM-based pipeline.

total positive detections, and recall represents the classifier's sensitivity with the ratio of true positive detections to the total *actual* positive observations. Accordingly, our classifier can identify different left-turn settings with high performance. Note that the number of instances of *unprotected* settings takes a large portion in our dataset due to its popularity in real-world, our classifier is not biased as evidenced by the performance metrics in Fig. 4.13. This shows the model trained on the augmented (and thus balanced) dataset is not biased and our LSTM framework successfully learns the feature(s) for differentiating left-turn protection schemes.

`TurnsMap` errs on the safe side of misclassifications to mitigate the impact of erroneous classifications. Specifically, it has a *lower* rate of misclassifying unprotected left turns as protected ones than that of classifying protected left turns as unprotected ones. This means that while identifying protected left turns with good accuracy, `TurnsMap` achieves better performance in identifying the unprotected left turns.

The performance of `TurnsMap` is essential for many applications, such as finding the safest left turns for novice drivers and route planning for self-driving cars, selecting a road with the minimum number of unprotected left turns using `TurnsMap`.

Now, we examine our classification performance based on the unbalanced dataset. To evaluate the classifier's performance while varying the threshold (i.e., cut-off probability), we first tested the performance and report the Receiver Operator Characteristic (ROC).

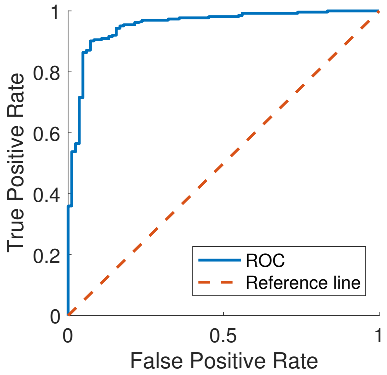


Figure 4.14: ROC curve.

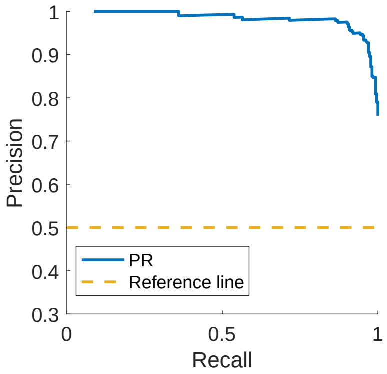


Figure 4.15: Precision-recall (PR) curve.

Then, to evaluate whether performance of protected left turns detection is not dwarfed due to the imbalance of the current dataset, we also reported the Precision-Recall (PR) curve to examine if precision captures the number of *false positives*.

As shown in Figs. 4.14 and 4.15, both the ROC and PR curves show the efficacy of our classifier. Specifically, both curves have significantly better performance than their corresponding reference lines (i.e., random guess). Moreover, the area under curve (AUC) for ROC and PR curves is 0.954 and 0.926, respectively.

#### 4.7.1.2 Comparison with Other Machine Learning Algorithms

Our model's performance is compared with the models trained by other learning algorithms based on the augmented dataset, and the *averaged* performance metrics are plotted in Fig. 4.16 where we tested a standard RNN, which is known to have good performance for sequential data [142]. One of its key differences from LSTM is that the standard RNN cell lacks gates (e.g., forget gate) for learning the sequential dependencies that may have varying (e.g., short- or long-term) timespans. We also compared with kernelized support vector machine (SVM), and random forest (RF). Specifically, the RNN uses the same hyper-parameter setting as our LSTM network. We used 200 decision trees for the RF algorithm. Note that we used the same training data setting for testing different algorithms. That is, the performance of RF and SVM with feature vector optimization is

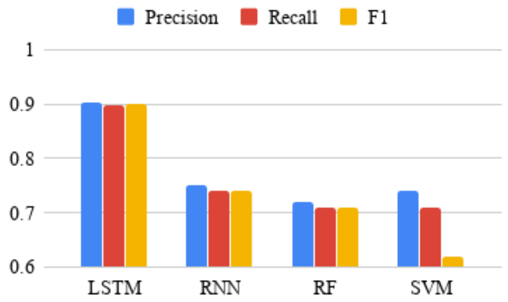


Figure 4.16: Comparison between LSTM and other machine learning algorithms.

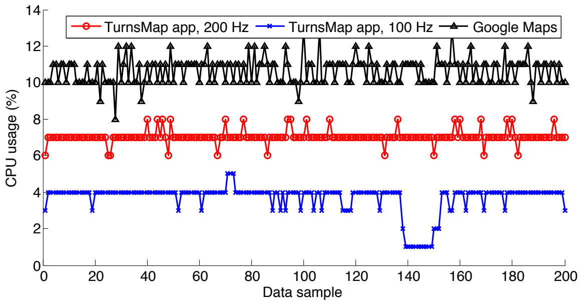


Figure 4.17: CPU usage of `TurnsMap` app.

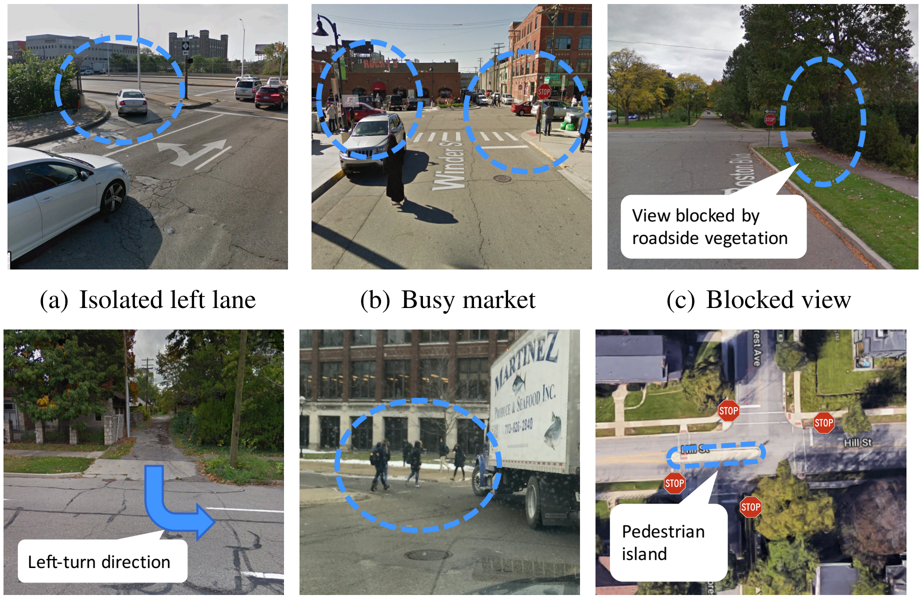
not the focus of this work.

For `TurnsMap`, LSTM outperforms the others in all metrics considered. As discussed in Sec. 4.6, the advantages of LSTM over others is its capability of capturing dependencies through time. For example, RNN cells are susceptible to the vanishing gradient decent problem, making it very hard to capture the dependencies in time-series data. SVM has a low average F-1 score, because it misclassified many unprotected hotspots as protected ones while showing good performance in detecting protected hotspots.

### 4.7.2 Overhead of the Data-collection App

Mobile devices are resource-limited and susceptible to power and/or computation-hungry apps. A high overhead of the data-collection app could consume lots of the mobile device's resources, thus discouraging use of `TurnsMap`. We have highlighted the data-collection app's generation of low network traffic in Sec. 4.4. Here we test its additional CPU usage and energy consumption on the users' mobile devices.

We installed our data-collection app on a Google Pixel phone that runs Android 7.1.2 Nougat OS. We experimented with two sampling rates of IMU sensor: 200Hz and 100Hz. We used the 100Hz sampling rate for most of time and also tested the 200Hz sample rate to study the maximum overhead incurred to users' smartphones.



(a) Isolated left lane (b) Busy market (c) Blocked view

(d) Exit w/o prot. (e) Crossing student (f) Pedestrian island

Figure 4.18: Real-world insights from TurnsMap result.

Since TurnsMap can be used as a collaborative add-on functionality for existing navigation apps (we will elaborate on this in Sec. 4.8), we also measured the overhead of a popular navigation app (i.e., Google Maps) and compared it with our data-collection app. Fig. 4.17 shows the CPU usage overhead of TurnsMap. It consumes an average of 7% and 3.7% CPU for 200Hz and 100Hz sampling rate, respectively, whereas Google Maps consumes 11% CPU. For the energy consumption, we measured the current drawn by the Pixel phone. It shows that the data collection app consumes only 45mA with IMU sensor sample rate of 100Hz, whereas Google Maps consumes 727mA. That is, the extra energy consumption is only 6.2% of the navigation app. This low overhead comes from the fact that the data collection activity does not use any power-hungry sensors (e.g., camera) and operates in background mode.

### 4.7.3 Real-world Insights from TurnsMap

TurnsMap can accurately differentiate the protection settings of different left turns. In the testing phase of TurnsMap (i.e., classifying testing data with the thus-learnt model), we have gained some interesting insights (Fig. 4.18).

#### 4.7.3.1 Protected.

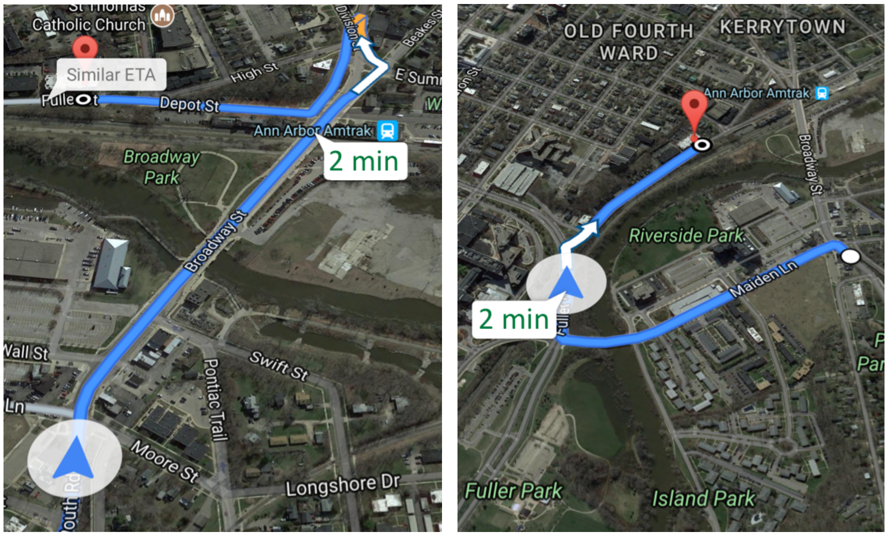
One of the left turns with protected settings is shown in Fig. 4.18(a) — an isolated left-turn *road* in an urban area. The driver can use this dedicated road for left turns, a complete isolation from other traffic and pedestrians.

#### 4.7.3.2 Unprotected.

Fig. 4.18(b) shows a busy market in an urban area where left-turning cars are likely to be blocked by a large crowd of people and street stalls. Fig. 4.18(c) shows an intersection enforced by a 2-way stop sign where the view of right-hand-side road is blocked by lush vegetation. Fig. 4.18(d) shows an intersection without any protection, which is also a joint spot of a narrow road with a major road (with a speed limit of 45mph). Such a left turn is very risky because the left-turning driver needs to pick up the speed quickly while paying attention to fast-approaching cars from the right-hand side. Fig. 4.18(e) shows an intersection with all-way stop sign located inside of a campus. Especially during inter-class time (e.g., 10:30am on weekdays), many students travel through this intersection to get to other classrooms, thus increasing the possibility of causing frequent conflicts between left-turning cars and pedestrians. Fig. 4.18(f) shows a confusing layout in a suburban area, where each entrance of the intersection is enforced by a stop sign, but there is a pedestrian island in the middle of the street. This blockage in the middle of the road requires drivers to be extra careful when turning left.

## 4.8 Use Case and User Study

Since left-turn enforcement is critical for driving safety, `TurnsMap` has potential for use in various app scenarios, such as navigation systems for both human- and self-driving cars,



(a) w/o `TurnsMap`

(b) w/ `TurnsMap`

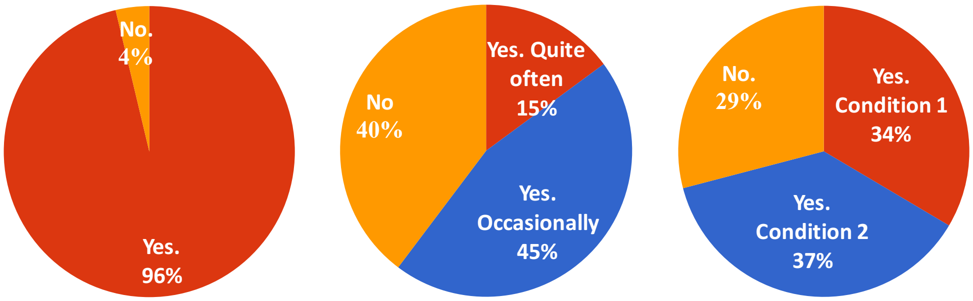
Figure 4.19: Exemplary application. Here, (a) and (b) compares the usage of a navigation app w/ or w/o `TurnsMap`'s capability.

and reporting unprotected left-turn hotspots to the local transportation department. In this section, we focus on assessing `TurnsMap`'s utility in navigation systems.

In existing navigation systems (e.g., Google map), one of the key limitations is the lack of safety assessment (as highlighted by Waze [18], finding dangerous left turns) of different routes. In fact, route/trip planning selects the route by minimizing the estimated time of arrival (ETA) [23]. As the result, the navigation system may generate a route with many risky unprotected left turns, as shown in Fig. 4.19(a).

One of `TurnsMap`'s exemplary application is augmenting existing navigation app with left turn setting information at intersections. That is, with the left-turn information discovered by `TurnsMap`, `TurnsMap` can provide this information with the collaborating navigation app to plan a route that also considers left-turn safety at intersections — which is not yet achieved functionality by using state-of-the-art approaches. Specifically, users of a navigation app can enable unprotected left turn avoidance in the setting of the app. Thus, the app can suggest an alternative route, as shown in Fig. 4.19, which has safer turning maneuvers (e.g., right turns).

To fully analyze user's satisfaction on this functionality, we first conducted a survey



(a) Question 1 (b) Question 2 (c) Question 3

Figure 4.20: User study of the exemplary app. Here, (a), (b), and (c) show distributions of participants' responses to our survey questions respectively.

of users' expectations of a navigation system that has the unprotected left-turn avoidance. Specifically, we recruited 564 participants from Amazon Mechanical Turk. Each participant needs to be a legitimate driver in the US to participate in our study. To assess people's expectation of this functionality, we asked the participants the following questions:

**Q1.** Can you differentiate left-turn enforcements?

**Q2.** Have you ever experienced dangerous/difficult unprotected left turns recommended by a navigation app?

**Q3.** If there is a navigation app that can help you avoid those risky left turns, would you use it?

Fig. 4.20 summarizes the survey results. For Q1, 96% of the participants can differentiate between protected and unprotected left-turns, indicating that most of our participants are aware of the protected/unprotected settings. For Q2, 60% of the participants experienced risky/difficult unprotected left turns recommended by the navigation app. Notice that 15% of the participants *often* encountered such left turns in the routes recommended by the navigation app. For Q3, 71% would like to use a navigation app that avoids unprotected left-turns while 34% prefer this functionality only if the alternative route has a similar ETA (Condition 1 in Fig. 4.20(c)). Surprisingly, *37%*

indicated that if driving safety can be enhanced, they won't mind taking an alternative route with a prolonged ETA (Condition 2 in Fig. 4.20(c)), i.e., safety is more important to a large portion of navigation app users. In summary, the avoidance of unprotected left turns is highly desired for navigation systems.

## 4.9 Limitation and Future Extension

Although TurnsMap can achieve high accuracy in differentiating protected and unprotected left-turn settings, there remains room for improvements, two of which are highlighted below.

### 4.9.1 Adapting TurnsMap to the Time-of-day

Currently, TurnsMap is trained based on driving data that is mostly collected during day-time (i.e., most of our training data is collected during 7:00 –18:00. Also, some intersections are equipped with advanced traffic regulation systems [150] that change the left-turn protection setting according to different times-of-day to accommodate the varying traffic conditions. The intersections that allow for flexible left-turn settings need both regular traffic light (i.e., circular light) and dedicated left-turn light (i.e., a green left-oriented arrow). For example, to maximize the traffic throughput during the morning rush hour, the intersection can use the regular traffic light to regulate left-turning traffic; during hours with moderate traffic, it uses the protected traffic light to enhance safety at the expense of traffic throughput. The detailed information (e.g., location and the specific left-turn setting) of these flexible traffic signals would be useful for various parties (drivers, navigation app developers). However, this information is unknown to the public.

TurnsMap can adapt to this sophisticated scenario by learning different classification models for different times-of-day. As shown in Fig. 4.21, 4 times-of-day can be 6:00 — 10:00, 10:00 — 16:00, 16:00 — 20:00, and 20:00 — 6:00 (the next day). Note that the first and third time segments denote the two usual daily rush hours. Based on the segmentation

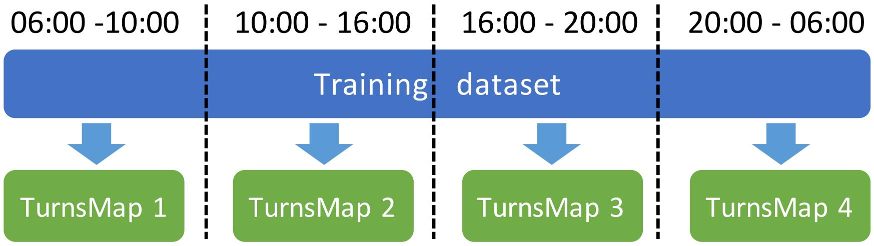


Figure 4.21: Illustration of formulating TurnsMap for different time-of-day.

of time, one can divide the training data using the timestamp of data collection and train the corresponding TurnsMap for each time segment.

### 4.9.2 Scalability of TurnsMap

We test TurnsMap by using the driving data collected from the Detroit metropolitan region in the U.S. that covers various on-road environments (e.g., suburban and urban areas). Moreover, since TurnsMap harvests interruptions (as elaborated in Sec. 4.5), the root cause of risks at intersections, TurnsMap can be used to enhance driving safety at large scale. Specifically, TurnsMap can be used in cities/regions that have the same/similar traffic regulations. For cities/regions with different traffic regulations, the current trained model of TurnsMap can help facilitate data collection during the training phase. In particular, the app developer or local transportation department could apply the transfer learning method [177] to use the existing model as a pre-trained model.

## 4.10 Related Work

### 4.10.1 Detecting the Driving Behavior

One of the major tasks of TurnsMap is to accurately detect the driver's maneuver based on mobile IMU sensor readings. Recently, there have been a growing number of research efforts to meet similar goals. For example, Yang *et al.* [174] used the smartphone accelerometer to detect if the phone is at the driver's seat. V-Sense [102] uses smartphones' built-in IMU sensors to detect various steering maneuvers. BigRoad [143] extracts the steering wheel angle from a smartphone's gyroscope. Researchers have also investigated the accuracy of using a mobile IMU sensor for detecting lane-change maneuvers [157]. One of `TurnsMap`'s novelties is to detect interruptions (e.g., caused by crossing pedestrians and/or oncoming cars), a unique feature that can be used to derive the left-turns information at intersections.

### 4.10.2 Inferring the Road Information

Left-turns information is critical to driving safety, but the traffic databases maintained by local governments (e.g., data.gov [43], such as Open Data Portal [33]) lack this information. The collaborative mapping database, OpenStreetMap, has recently proposed a new feature to cover traffic light information [36] at intersections. However, this new feature is still in its infancy (i.e., under the community's review) and may take years to become practical. Even after receiving an approval, it will require a long time for the contributors (i.e., mappers) to collect the data due to the limited number of contributors.

To acquire traffic light/signal information at intersections, the corresponding industry has been devoted to the collection of left-turns information by taking images of intersections with heavily-instrumented mapping cars. For example, high-definition mapping services (Google Street View, here [20], TomTom [42]) use dedicated mapping cars to collect street images and derive intersection information automatically by using computer vision algorithms. However, this approach requires a significant investment of infrastructure, thus limiting its scalability, e.g., too expensive to use in under-developed or developing regions.

There have been proposals to detect intersection information with cameras in more efficient ways, but they suffer the natural limits of image data due to a variety of outdoor conditions [111]. Besides, placement of various traffic signals/lights make it hard for onboard perception systems to ensure that the intersection information is captured by using a limited number of cameras [115].

Aly *et al.* proposed Map++ [94], a crowdsourcing platform for mining traffic sign information from sensor data collected from mobile devices. However, Map++ detects only coarse-grained road information, e.g., existence/absence of traffic light. TurnsMap extracts and uses the spatio-temporal features from IMU sensor data to infer the specific protection type at left-turn hotspots — a safety-critical information unachievable with Map ++.

Recently, a traffic pattern analysis with mobile crowdsensing is gaining popularity due to its potential for benefiting smart cities. SmartRoad [134] uses mobile GPS data to determine whether or not traffic lights and/or stop signs are present. However, none of these existing studies explored left-turns information at intersections, albeit its importance to driving safety.

Unlike existing schemes, we focused on the left-turns information, a safety-critical yet unavailable data which can be efficiently collected by TurnsMap's crowdsensing. By crowdsensing mobile IMU sensor readings, TurnsMap can collect this information in an efficient, scalable manner.

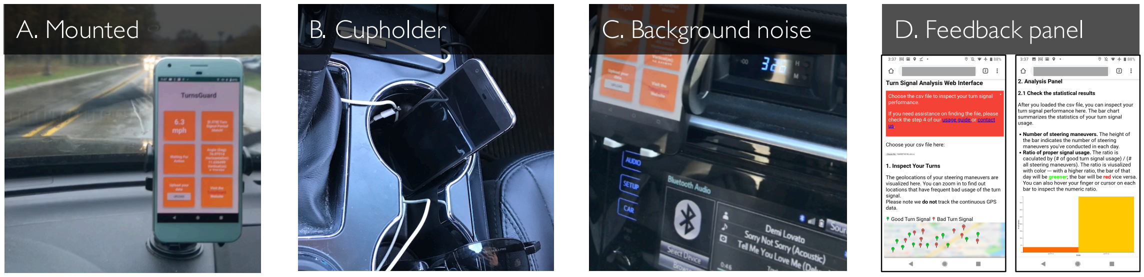
## 4.11 Conclusion

TurnsMap is a novel way of utilizing crowdsensing mobile IMU data and deep learning for enhancing driving safety. It can infer the left-turn protection settings on road segments with high accuracy. To train and evaluate TurnsMap, we constructed a real-world dataset. TurnsMap is empowered by two key enablers: a novel data mining engine for extracting left-turn hotspots and a deep learning-based pipeline for learning the model for classifying left-turn enforcements. The left-turns information uncovered by TurnsMap is essential for driving safety and can benefit many vibrant apps in the automotive and transportation ecosystem.

# CHAPTER V

# TurnsGuard: Smartphone-based Profiling of Driver's Attentiveness to Turn-Signal Usage

Making turn(s) and/or changing lane(s) without giving turn signals is a dangerous driving behavior that causes a variety of traffic accidents. However, there does not exist any good way of detecting and analyzing a driver's turn-signal usage and then reminding him/her if s/he made a turn without giving a turn signal. To mitigate this deficiency, we present `TurnsGuard`, a practical and scalable turn-signal profiling tool only with commodity smartphones. `TurnsGuard` can be implemented as a smartphone app to monitor the vehicle's steering maneuver with phone sensors and detect the existence/absence of turn signals by analyzing the acoustic feature picked up by the phone's microphone. `TurnsGuard` works *locally* on the driver's mobile device, thus enhancing `TurnsGuard`'s usability while preserving the driver's privacy. The evaluation started with testing on 10 different vehicle types in real-world settings, `TurnsGuard` can achieve 94% precision in turn signal detection. According to our field study with a group of participants, `TurnsGuard` can effectively profile and characterize the drivers' turn-signal usage, thus enabling an essential approach for understanding driver's attentiveness.



3:37 [redacted]

Turn Signal Analysis Web Interface

Choose the csv file to inspect your turn signal performance.

If you need assistance on finding the file, please check the step 4 of our usage guide or contact us.

Choose your csv file here:

1. Inspect Your Turns

The geolocations of your steering maneuvers are visualized here. You can zoom in to find out locations that have frequent bad usage of the turn signal.
Please note we **do not** track the continuous GPS data.

3:37 [redacted]

2. Analysis Panel

2.1 Check the statistical results

After you loaded the csv file, you can inspect your turn signal performance here. The bar chart summarizes the statistics of your turn signal usage.

- **Number of steering maneuvers.** The height of the bar indicates the number of steering maneuvers you've conducted in each day.
- **Ratio of proper signal usage.** The ratio is caculated by (# of good turn signal usage) / (# all steering maneuvers). The ratio is viusalized with color --- with a higher ratio, the bar of that day will be greener; the bar will be red vice versa. You can also hover your finger or cursor on each bar to inspect the numeric ratio.

Figure 5.1: We designed, implemented and tested a smartphone-based tool (TurnsGuard) for profiling a driver's inattentive steering maneuvers, i.e., turning or changing lanes without giving turn signals. A, B, and C demonstrate TurnsGuard's robust performance under different usage contexts — mounted, cupholder, and under background noises (e.g., music), respectively. Please refer to our Video Figure to see how TurnsGuard works in the real-world. D shows the online feedback panel (elaborated in Sec. 5.4) for contextualizing the detection results and arousing the driver's attention.

## 5.1 Introduction

Steering maneuvers imply a rich set of information that can reflect the level of safety and attentiveness of the driver. In particular, fast cornering turn signals or blinkers — installed on the left and right front and rear corners of cars since the first introduction of cars in 1909 [10] — have been a vital safety feature required by law worldwide. The turn signal enables communication between cars (or drivers) which can effectively alert nearby vehicles of the ego-car's intended movement and is thus crucial information for avoiding potential traffic accidents. In fact, turn signals have been recognized as the *most fundamental* critical safety protection [73], and their use by the drivers is enforced by law (e.g., U.S. Department of Transportation) and the traffic enforcer.

Unfortunately, many drivers make turns and/or change lanes often forgetting to give turn signals. Studies and news coverage in psychology and driving behaviors have shown [56,83], forgetting to use turn signals has severe repercussions: if the driver neglects to use turn signals, s/he will eventually *forget* to use the signal when it is really needed. According to a recent field study conducted by the Society of Automotive Engineers (SAE), 48% of the drivers in the U.S. do not give turn signals when they change lane;

more than 25% of them fail to give signals when they make turns [15]. As a result, drivers' failure to use turn signals alone accounts for more than 2,000,000 accidents annually [64]. Thus, enhancing the driver's attentiveness to use of turn signals is vital not only for the safety of individual drivers, but also for the safety and efficiency of the entire traffic system.

To conduct an in-depth study of drivers' turn-signal usage and enhance their attentiveness, we need a practical and scalable tool for detecting their (un)use of turn signals. In particular, the tool should be capable for both micro behavioral analysis (e.g., an individual driver's attentiveness to turn signals) and macro sociological study (e.g., how drivers' turn-signal usage may impact traffic safety on a large scale). Moreover, this tool should also be privacy-preserving, i.e., balancing needs between the system performance and privacy concerns. However, to the best of our knowledge, there does not exist any practical approach/tool that can meet these requirements.

To meet this pressing need, we propose `TurnsGuard`, a smartphone-based system for detecting and analyzing the drivers' turn-signal usage. Specifically, `TurnsGuard`'s sensing is designed and built with two key modules for detecting the turn-signal usage only with commodity smartphones carried by the drivers.

**Steering Detection Module.** `TurnsGuard` uses a novel way to detect a turning maneuver without using any vision/camera data. In particular, the collection of vision information requires mounted camera(s) that must persistently focus on the road — a requirement that may make `TurnsGuard` unusable. Moreover, the detection performance by using image data can be easily distorted by various environmental conditions, such as the road covered with snow and/or poor lighting. To avoid these difficulties, `TurnsGuard` utilizes motion sensors (e.g., gyroscope and accelerometer) on commodity smartphones for detecting turn maneuvers. Our steering detection also harvests sensor calibration to overcome different phone posture (e.g., mounted vs. cupholder as shown in Fig. 5.1 A and B).

**Turn-Signal Detection Module.** `TurnsGuard` uses the smartphone's microphone as peripheral sensing modality for detecting consecutively "clicking" turn-signal sound. Although a turn signal can be detected accurately by accessing the vehicle's CAN bus [175], the *de facto* in-vehicle network that connects various modules/ECUs inside a car, it has several usability restrictions in the real-world. First, due to the ever-increasing security concerns of car hacking [59] via the CAN bus, car-makers have been working to restrict the access of in-vehicle data, e.g., by shutting down [69] and/or encrypting [58] the onboard diagnostic (OBD-II) [54] port. Second, since the CAN data packet frame format varies with car-makers and/or car-models, interpreting or reverse-engineering the CAN data would be very difficult without "translation dictionaries" which are proprietary to car-makers. In contrast, `TurnsGuard` does not require any additional accessory or proprietary information, thus enhancing its usability and deployability.

The detection of a turn signal by recognizing its clicker sound, however, can be challenging because 1) the turn-signal sound can be easily distorted by various environmental noises, such as mechanical vibrations, human voice, and loud music; 2) although the clicking sound is designed to be audible, the sound configuration — e.g., amplitude and frequency — may vary with car models and makers. To overcome these challenges, we have implemented an unsupervised turn-signal recognition pipeline that allows `TurnsGuard` to be used *locally* on the user's smartphone. In the first phase, `TurnsGuard` makes a short-duration recording of the user's vehicle's turn-signal sound. This is a *one-time* effort for each car-model and can be done within a minute even for inexperienced users of `TurnsGuard` according to our field study (to be elaborated in Sec. 5.4). The second phase matches, in real time, the sound of the vehicle's turn signal to automatically detect the turn signal. In the third and final phase, `TurnsGuard` is customized for the driver's vehicle. To evaluate the performance of `TurnsGuard`, we recruited volunteers to drive their own cars with smartphones as naturally as possible.

We achieved robust steering and turn-signal sensing by using commodity smartphones without any extra physical vehicle connections or accessories, opening a wide range of new channels that could unlock a wide range of automotive applications. To further support the usability of TurnsGuard, we employ a front-end/back-end co-design. As we will elaborate in Sec. 5.3, we integrate the sound sensing into a front-end app, whereas the back-end harvests the cloud support for managing and contextualizing user's data. Based on our on-road user study (Sec. 5.5), the information of turn-signal (un)usage can be a cornerstone for studying driving behavior and developing applications thereof. The statistical report of turn-signal usage can also be used for educating/training/improving drivers on safety, e.g., we demonstrate (in Sec. 5.5) an on-road use case that contextualizes the driver's turn-signal usage for arousing his/her attention to driving (as shown in Fig. 5.1 D). Overall, TurnsGuard is shown to sense and determine the driver's turn-signal (un)usage with commodity mobile devices, thus enabling a wide range of mobility applications.

## 5.2 Related Work

### 5.2.1 Driving Behavior Analysis

Recently, industry has started using a driving behavior analysis as an effective way of enhancing driving safety. In particular, developers use the onboard diagnostic devices (OBD-II) dongle and/or smartphone apps for monitoring the users' driving behavior. For example, automotive insurance companies (e.g., Progressive and StateFarm) have been working on a fast-emerging program called *Usage-Based Insurance* (UBI) [78] for adjusting insurees' premium based on their driving behavior. To enroll in UBI, drivers need to install the dongle provided by the insurer for monitoring the driving behavior.

Although this program has been attracting a substantial number of participants, the existing driving behavior analysis can only detect a very limited type and number of driving behaviors. In fact, current UBI products are only able to detect two driving behaviors: speeding and hard braking. This limited behavior set is neither comprehensive nor able to accurately reflect the driver's safety and/or attentiveness to safety-related maneuvers. For example, hard braking for dodging crossing pedestrians and/or animals is a safe maneuver. It would be very useful and safe if the driving behavior analysis service can detect the failure to use turn signals, which is a well-known dangerous driving behavior closely related to inattentive drivers.

### 5.2.2 Monitoring Driving Behavior with Mobile Devices

Researchers have been exploring the feasibility of assessing the driving behavior with accessible devices such as smartphones. Hong *et al.* [133] demonstrated use of both a smartphone and special hardware (with the combined cost around $200) for identifying aggressive drivers. However, their system cannot profile the turn-signal usage. Zhang *et al.* [180] and Chen *et al.* [103] proposed different methods of using a smartphone's motion sensor data for characterizing the driving style. They demonstrated the potential of extracting rich information from the driving data for supporting various automotive applications. Recently, Chen and Shin [104] have used the crowdsourced smartphone's data to classify intersection protections (e.g., whether an intersection has left/right turn protection) — a safety-critical information for driving. Chen *et al.* [102] and Liu *et al.* [144] briefly mentioned the concept of using smartphones for detecting the usage of turn signals. However, none of these works have designed and tested a practical system for profiling the usage of turn signal.

In summary, prior work demonstrated a significant potential for using smartphones as a scalable and accessible modality for safe driving. However, none of them has considered the turn-signal detection problem, despite its importance. As we will elaborate in the remainder of this paper, the turn-signal (un)usage data and its analysis enabled by `TurnsGuard` can open a new area of analyzing driving behavior and introduce/enhance various applications.

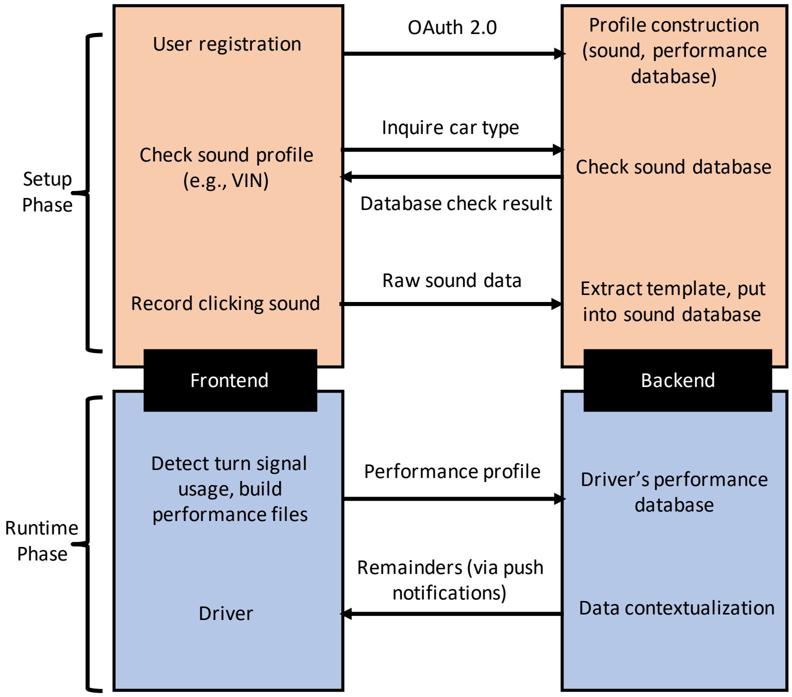

Setup Phase
User registration
OAuth 2.0
Profile construction (sound, performance database)
Check sound profile (e.g., VIN)
Inquire car type
Check sound database
Database check result
Record clicking sound
Raw sound data
Extract template, put into sound database
Frontend
Backend
Runtime Phase
Detect turn signal usage, build performance files
Performance profile
Driver's performance database
Remainders (via push notifications)
Driver
Data contextualization

Figure 5.2: The front-end and back-end design of `TurnsGuard`.

### 5.2.3 Improving Driving Safety

How to encourage safe driving effectively has been an important research topic in the HCI community. Chin *et al.* [105] proposed use of social media for incentivizing drivers to avoid dangerous driving, i.e., speeding, abrupt braking, and making sharp turns. Tanaka *et al.* [163] showed three different interactive channels for enhancing safe driving for elders. Their system alerts the driver if a dangerous behavior is detected.

To the best of our knowledge, none of prior studies investigated the safety and behavioral implication of turn-signal usage patterns. We conjecture the primary reason for this is the lack of a scalable way of detecting turn signals. `TurnsGuard`'s main goal is to remove/narrow this gap and demonstrate how its detection of turn-signal (un)usage can be detected for enhancing driving safety.

## 5.3 System Design of TurnsGuard

The goal of TurnsGuard is to provide a *usable*, *scalable*, and *privacy-preserving* tool for transportation stakeholders to assess the drivers attentiveness based on their steering maneuvers. In this section, we first articulate the three design goals. Next, we present the usage model of TurnsGuard enabled by the front-end/back-end co-design. Finally, we elaborate the design of our turn signal sensing scheme.

**Usability.** We achieve characterization of real-time steering maneuver and detection of turn-signal (un)usage with the drivers' smartphones. This allows the stakeholder to achieve flexible and real-time accessibility of the driver's behavioral data, an essential enabler for many useful automotive apps, such as auto insurance, ride-sharing, and parental monitoring.

**Scalability.** There are two key varying factors that hinder large-scale detection of steering maneuvers with smartphones: different sensing device models and different vehicle models. Specifically, different smartphone models may have varying configurations of the sampling rate and sensing hardware, and different car types usually have different spectral features (e.g., period, tone, and frequency) of the turn-signal sound. TurnsGuard unifies the sensor usage on different smartphones by using a novel steering sensing scheme called *V-Sense* [102]. To achieve robust turn-signal sensing for different car types, the back-end support of TurnsGuard crowdsources the turn-signal sound from users in the setup phase (to be elaborated in Sec. 5.3.1) to construct a comprehensive sound database. This allows users to reuse different vehicles' sound templates with ease.

**Privacy-preservation.** Protecting end-users' privacy while retaining the usability and scalability is a key requirement for TurnsGuard. We implemented two key privacy protection features to meet this goal. First, TurnsGuard's turn-signal detection module only uses the microphone during the vehicle's turn. The subsequent sound data analytics (as elaborated in Secs. 5.3.2–5.3.4) are done efficiently on the smartphone to analyze the recorded sound snippet, which is discarded immediately after the analysis. That is, no sound data is stored locally or uploaded to the server. Second, TurnsGuard's driver assessment uses access control on the back-end resources while sharing the sound template.

### 5.3.1 Usage Model of TurnsGuard

As shown in Fig. 5.2, there are setup and run-time phases to use TurnsGuard.

#### 5.3.1.1 Setup Phase

The first-time usage of TurnsGuard front-end app will guide the user through the three-step setup phase. In the first step, the user needs to register his account with Google OAuth 2.0 [76] to construct his profile, including the sound profile of his car and performance database. Specifically, the sound profile is a light sound data extracted from the turn-signal sound of the user's car. This file is essential for the real-time sound analysis as elaborated in Sec. 5.3.4. The performance database includes the metrics for assessing the driver's steering maneuver. Each entry of the database is constructed with four elements:

$$\begin{split}\{time\_stamp, geo\_cooridnate, steering\_type,\\ sound\_usage, latitude\_acceleration\}.\end{split} \tag{5.1}$$

Then, the back-end will inquire of the driver's car type, which can be accurately identified with the vehicle's VIN number. This step is to checking whether the sound template of the user's car is already in the back-end sound database. If TurnsGuard already has the required sound template, the user can skip the third step.

In the third step, the user needs to provide a raw recording of his car's turn-signal sound. The users will be asked to collect a 10-second recording of their vehicles clicking sound in a quiet setting. To set up the recording environment, the participants are instructed to put the vehicle into ACC mode (e.g., accessory mode with electronic devices on and engine off), turn off radio, and close all doors. Next, in order to record clear sounds with low disturbances, the users are instructed to place their phones on the dashboard before starting the recording. Then, the front-end app will automatically send the recorded sound clips to the back-end for extracting the template turn-signal sound and customizing the corresponding TurnsGuard. Subsequently, the template will be sent back to the smartphone to finish the customization of the user's app. The sound collection step enables TurnsGuard to crowdsource sound templates of different vehicles in an efficient way. This is a one-time effort for each user, which is an acceptable and easy task according to our interviews of our on-road user study (elaborated in Sec. 5.5). The extracted template (i.e., sound model) will be sent back to the user's app via HTTP post in TXT format. Now, the app is fully workable locally on the smartphone for the run-time phase.

#### 5.3.1.2 Run-time Phase

TurnsGuard implemented two features to mitigate the front-end app's distraction. First, it automatically detects whether the user is driving a car by using Google Context Fences API [67]. Specifically, this API detects the smartphone's GPS speed to differentiate transportation modes, e.g., biking and driving. This feature enhances the usability of TurnsGuard, as the user need not manually start and stop the app. Second, to allow multi-tasking and lower the energy overhead of the screen activity, our app uses Android Foreground Service [65] to enable the background mode.

At run-time, TurnsGuard app detects the steering maneuver and the turn-signal usage (see Sec. 5.3.2) and interacts with the user via real-time alerts and a comprehensive driving report. To alarm un-signaled turns in real time, TurnsGuard front-end can make a pop-up notification to the driver without sharing any data sharing and/or interacting with the server. By accumulating the user's driving data, TurnsGuard back-end can also provide a comprehensive driving report to the app users and other eligible stakeholders (e.g., insurance provider, local transportation department). The microscopic analysis of each

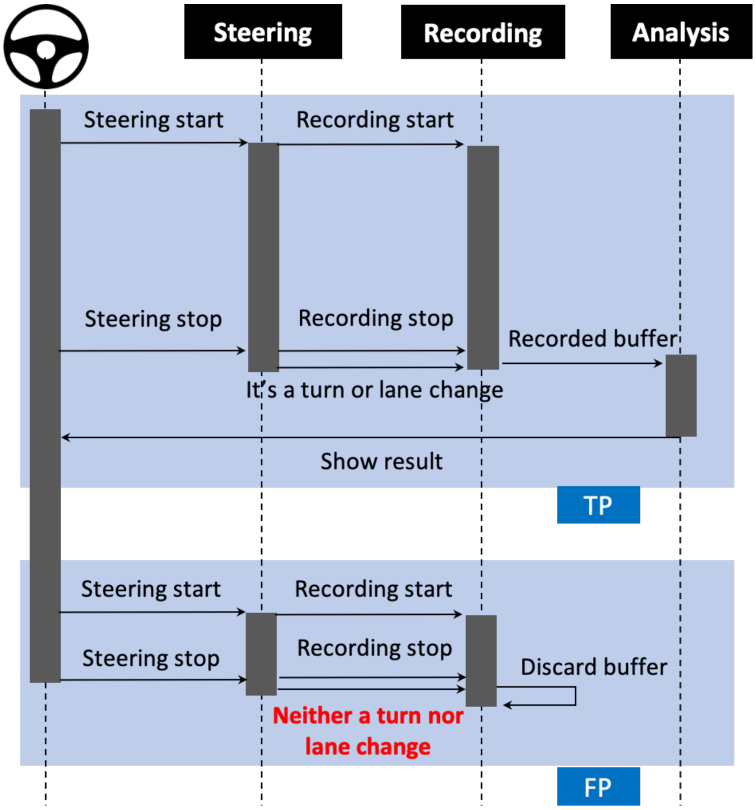


Figure 5.3: The workflow of `TurnsGuard`'s front-end.

driver's turn-signal usage can benefit the driver's attentiveness, while using a macroscopic analysis of the aggregated behavioral data from a large number of drivers, stakeholders may gain insights into traffic problems on the roads.

### 5.3.2 Front-end Design

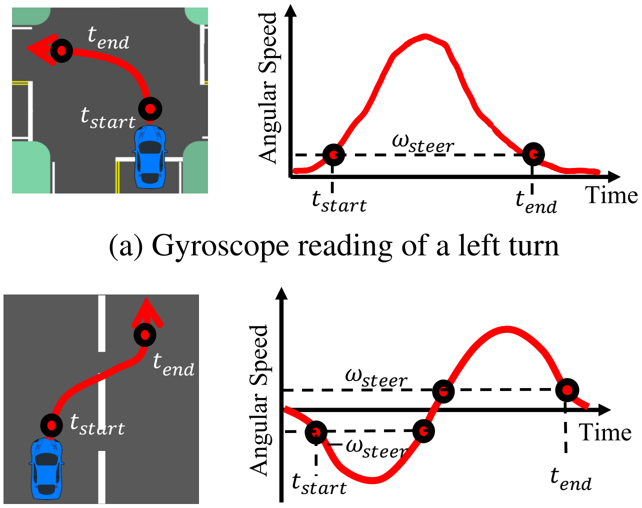
We employ a multi-thread design in `TurnsGuard` to enable seamless computation by organizing steering detection, sound detection, and sound analysis as different threads. Fig. 5.3 shows an operation sequence diagram, illustrating the workflow of `TurnsGuard`. Specifically, when a steering maneuver takes place, the steering detection thread will trigger the sound detection thread to activate the recording with microphone. Then, if a steering maneuver is a left/right turn or a lane change, i.e., true positive (TP), the recording thread will trigger the analysis thread to process the recorded sound data. Finally, the analysis result, i.e., whether a turn signal is used or not during this steering maneuver will be shown and/or recorded. This multi-thread framework is able to coordinate different operations, e.g., the vehicle starts steering while TurnsGuard is analyzing the sound snippet. Note that this framework can handle the false positive (FP) detection in a flexible manner without inducing high computational overhead, e.g., the pipeline will skip the sound analysis if the steering maneuver is neither a turn nor a lane-change.

The key advantage of this framework is that it only requires the sound snippet during a turn (whose time duration is around 3–5 seconds), thus preserving the user's privacy. Also, it enables the detection of turn signals to alert drivers in real time.

### 5.3.3 Detection of Steering Maneuvers

To minimize users' privacy concern and mitigate the workload on resource-limited mobile devices, the steering detection thread is implemented by using the smartphone's motion sensors, i.e., gyroscope and accelerometer. These sensors generate one-dimensional time-series data, which can be used for sensing the vehicle's angular speed and velocity during steering. We employ the sensor fusion of V-Sense II, and demonstrate a real-time data analytics pipeline for detecting a vehicle's steering maneuvers.

The key idea of V-Sense is illustrated in Fig. 5.4. To detect steering maneuvers, it analyzes the phone's gyroscope data to find the starting and ending points of each steering maneuver by using a threshold $\omega_{steer}$. This threshold is derived from natural driving maneuvers (e.g., from the change of the vehicle's heading) and is applicable to different mobile platforms. For example, when the car is turning left, the gyroscope reading first increases, and then decreases as shown in Fig. 5.4(a). So, the starting and ending points of the "bump" shape in the figure are denoted by $t_{start}$ and $t_{end}$, respectively. V-Sense can also detect a lane-change maneuver: after each $t_{end}$, V-Sense applies a *soft* time window to examine if there is a follow-on bump — the indicator for a lane-change — as shown in Fig. 5.4(b).



(a) Gyroscope reading of a left turn

(b) Gyroscope reading of a lane-change

Figure 5.4: Correlation between steering maneuver and IMU sensor.

For accurate detection of turns and lane-changes (*active* steering maneuvers), it is important to differentiate them from driving on curved roads (*passive* steering maneuvers). V-Sense achieves this by using the vehicle's horizontal displacement. That is, a vehicle traveling on a curved road often makes a much larger horizontal displacement than active steering maneuvers. Note that, to overcome different smartphone postures (e.g., mounted vs in the cupholder) V-Sense also performs coordinate alignment [181] for transferring the motion sensor's coordinate to the Earth's coordinate. This design allows robust steering detection despite the varying posture of the phone.

We have modified V-Sense to dynamically start and stop the sound detection thread. Specifically, when the turn detection module finds the start of a bump (i.e., $t_{start}$), the phone's microphone will be triggered to record the sound; the recording will be terminated upon detection of the end of a bump (i.e., $t_{end}$). If the steering maneuver is classified as a turn, the thus-recorded sound snippet will be passed to the sound analysis thread for further analysis.

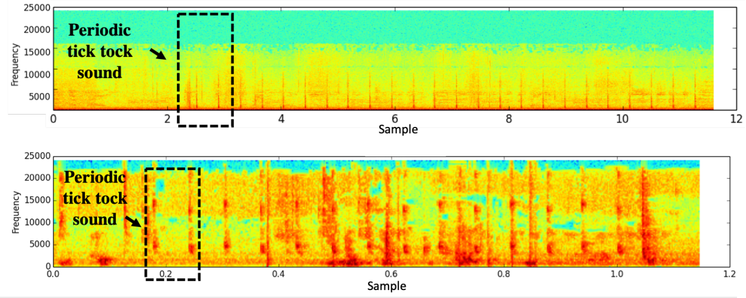


Figure 5.5: Spectrograms of Lincoln MKZ (upper) and Ford Explorer.

### 5.3.4 Detection of Turn Signals

Our sound analysis thread detects the existence of a turn signal by exploiting the periodicity of the sound signal. In essence, a vehicle's turn signal sound is a repeating clicking sound with the frequency and volume that humans can hear. Based on SAE J590b [51], the frequency of a turn signal ranges between 60 and 120 flashes per minute — each flash should also be associated with a clicking sound. To understand the statistical and spectral features of a turn signal sound, we show the spectrogram of two turn signal sound snippets from two different vehicles in Fig. 5.5. The recordings include the turn-signal sounds of two vehicles: (1) Lincoln MKZ's sound is recorded in a quiet environment, i.e., the car is stationary and the engine is off; (2) the recordings from Explorer contain a human speaking sound and engine noises. The spectrogram provides two insights. First, the clicking sound repeats at stable intervals. Its periodicity is a more robust feature than other acoustic features (e.g., formant position and spectral density), considering the high background noise in cars. Second, the main formant of turn signals lies between 3000 and 10000Hz. This observation inspired us to design a real-time sound analytics to detect the turn-signal sound by exploiting its periodicity.

The main challenge in designing the sound detection module is how to mitigate the effects of background noises, i.e., mechanical noise, human voice, and music sound. To tackle this challenge and enable `TurnsGuard` to operate in real time, we propose a light-weight periodic sound recognition pipeline powered by matched filtering and auto-correlation analysis. Specifically, our pipeline includes four steps: (1) formation of a template from the vehicle's turn-signal sound, (2) maximizing the signal-to-noise ratio (SNR) with the matched filter, (3) adaptive pattern matching with auto-correlation, and (4) checking the existence of sound periodicity. These steps are elaborated next.

#### 5.3.4.1 Formation of the Principal Clicking Sound

A representative template of the vehicle's clicking sound is needed to filter out noises and recognize the periodicity of sound signals. If the car type is new, the user needs to provide a sound recording of the turn-signal sound when the car is parked, without background noise, e.g., music and speaking sound. Our sound extraction first finds peaks of the turn-signal sound and then frames the clicking sound for each peak. This two-step approach allows us to form a sample set of 20 – 40 clicking sound snippets with 10-second sound recording (provided by users). Finally, we construct the principal clicking sound using Principal Component Analysis (PCA) [173] based on the sample set. The sound template will be formulated into a TXT file in the back-end service. Note that the back-end server archives (in the sound database) the sound template file for future use. Then, the server dispatch the template file to the user's smartphone via HTTP post. So far, we have investigated the principal clicking sound from 10 different vehicle types, includes compact (Corolla, Mini, 500X), mid-size (Accord, MKZ, Camry, Fusion, Q50), compact-size crossover (CRV), and mid-size SUV (Explorer). We present the corresponding extracted turn signal templates in Fig. 5.6. Here, the unified sampling rate is 16 KHz, the averaged duration of sound template is 25.14 ms (STD: 3.89 ms). The average file size of the resulting TXT files (from 10 different vehicle types) is 6,231 Bytes (STD 269 Bytes) — a lightweight sound template for identifying clicking sound in real time.

#### 5.3.4.2 Maximizing Signal Sound SNR

As mentioned earlier, the sound recorded during driving can be heavily distorted by the background noise, which causes low SNR of the signal sound, thus undermining the

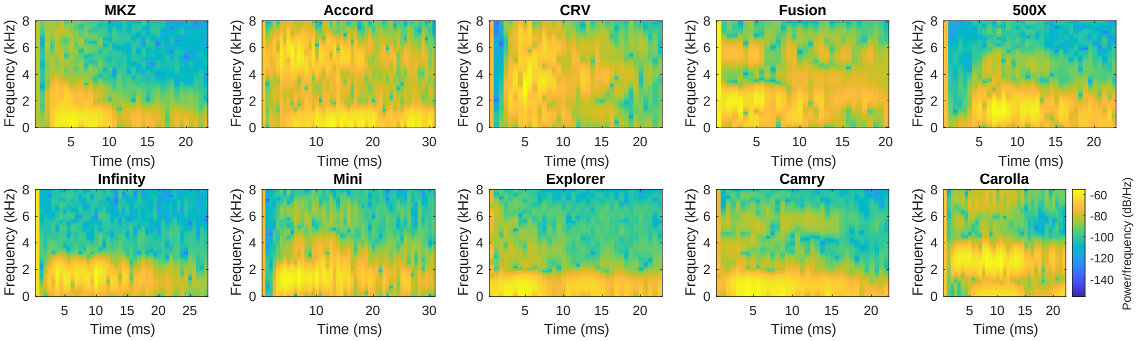


Figure 5.6: The spectrograms of the extracted templates of different cars' turn signal sounds.

detection result. To obtain a higher SNR of the signal, we harvest the discernible repetitive feature from the recorded sound snippet $S$ using a matched filter [167]. In essence, the matched filter is designed to *maximize* the SNR of the sound signal by computing the convolution of the signal with a reversed template.

Constructing a matched filter involves noise modeling and signal convolution. We model the noise $G$ inside the car as an additive Gaussian noise by designing the noise core as:

$$G = [2,1,0,-1,2] \circledast [-2,-1,0,1,2]. \tag{5.2}$$

We then convolve the reversed template (denoted as $T'$) with the modeled noise to construct the matched filter as:

$$h_m = G \circledast T'. \tag{5.3}$$

Finally, we calculate the maximal SNR by convolving the sound snippet $S$ with $h_m$:

$$S_h = S \circledast h_m. \tag{5.4}$$

Note that, by using the short time snippet and template, `TurnsGuard`'s design also mitigate the $O(|S| \cdot |h_m|)$ time complexity in the convolution step. Application of a matched filter can effectively substantiate the feature of interest — the *periodicity* of the sound snippet.

We benchmarked the matched filter's performance with real-world experimental data, i.e., recordings of turn signal sound inside of a 2011 Honda Accord under three different environment noise settings. A Google Pixel phone (embedded with a 50H01196 microphone) is used for the benchmark data collection. To quantify the environment noise level, we applied the sound pressure level (SPL) [53], the reference data is recorded by using the Pixel phone inside of the car without any signal and/or noise sound. Specifically, the quiet condition (SPL = 21.43 dB) only has turn signal sound; the moderate noise condition (SPL = 49.56 dB) includes the indling engine sound and background music; the noisy condition (SPL = 62.12 dB) includes engine sound while driving, and background music. Our matched filtering scheme is able to extensively boost the SNR under different noise settings. Specifically, the SNR increased from -24.78 dB to -5.67 dB under the moderate noise condition; whereas in the noisy condition, the SNR increased from -29.31 dB to -7.18 dB.

#### 5.3.4.3 Adaptive Pattern Matching

The goal of this step is to find the repetitive feature of the clicking sound. Our key enabler is based on finding the auto-correlation coefficient [158] of the filtered turn signal sound $S_h$. We use auto-correlation for detecting periodicity as it has been widely used (e.g., in music information retrieval) for the detection of periodicity in time-series signal data. In essence, the auto-correlation is a time-domain operation that convolves $S_h$ with itself iteratively using an increasing lag. This way, the resulting auto-correlation function (ACF) would show peaks at $\mathbb{N} * T$, where $T$ is the signal's period and $\mathbb{N}$ a natural number. We also optimized the auto-correlation calculation on smartphones, because the complexity of the brutal-force calculation of auto-correlation is $O(|S_h|^2)$, and hence is too computationally expensive to be used on mobile devices. To tackle this problem, we use the Fast Fourier Transformation (FFT) implementation of auto-correlation, which reduces the time complexity to $O(|S_h| \cdot log(|S_h|))$.

#### 5.3.4.4 Recognizing Sound Periodicity

As mentioned above, the ACF of $S_h$ shows peaks at integer multiples of its period. After computing ACF for $S_h$, we apply a peak-finding algorithm to ACF to extract $T$. The subsequent peaks in ACF show a descending order of magnitudes, since the overlaps of a signal with itself are in descending order. Therefore, the next global peak should be at $T \cdot f_s$, where $f_s$ is the sample rate of the sound recording. Note that it is a standard regulation that the turn-signal period of vehicles should range between 0.4 and 1 seconds. Because the period of a turn-signal sound consists of two similar clicks (e.g., tik-tok sound), the peak at position $T/2$ will often be recognized as the second peak in noisy environments. For detection consistency, we set the search area in $[0.2, 0.5]$ seconds to find if a peak exists at $T/2$. Let $P_{peak}$ be the position at which the second peak occurs, the period can then be calculated by

$$T_{detected} = 2P_{peak}/f_s. \tag{5.5}$$

Finally, we can conclude that $T_{detected}$ matches $T_{template}$ by checking if

$$|T_{detected} - T_{detected}| \leq \varepsilon, \tag{5.6}$$

where the bound $\varepsilon = 0.05$ second is predefined based on our experimental results. The detection of each steering maneuver will be formed as a CSV entry:

$$\begin{aligned}
&\text{"}Time\,stamp\text{"} : <long : UNIX\_TIME>, \\
&\text{"}Latitude\text{"} : <double : LATITUDE>, \\
&\text{"}Longitude\text{"} : <double : LONGITUDE>, \\
&\text{"}Steering\,type\text{"} : <int : TYPE\_ID>, \\
&\text{"}Turn\,signal\,usage\text{"} : <boolean : IF\_USED>
\end{aligned} \tag{5.7}$$

where the type ID 1, 2, 3, and 4 denote left turn, right turn, left lane change, and right

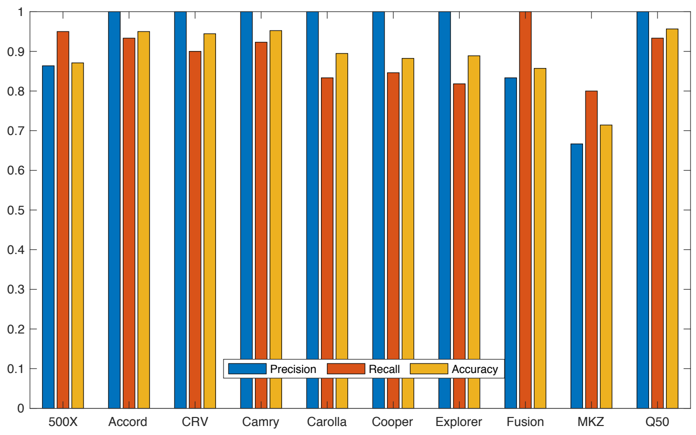
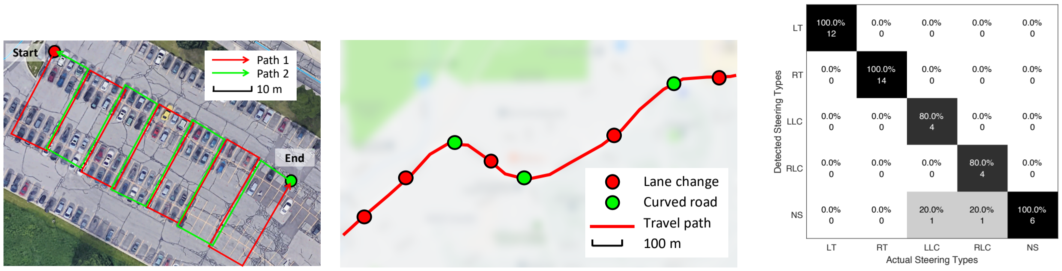


Figure 5.7: Statistical results: precision and recall of the sound detection module on different types of vehicle.

lane change, respectively. The driver can decide to upload each detection result in real time or accumulate detections locally in his smartphone. To support real-time alarming, if $IF\_USED = 0$, the app pushes a notification to alert the driver. The CSV entries are saved locally, and the driver can decide when to upload them to the cloud for a further analysis.

### 5.3.5 Performance of Clicking Sound Detection

To examine the practicability of our sound detection scheme cross different types of cars, we designed experiments that cover a wide range of turn signals from the 10 aforementioned car types. To evaluate TurnsGuard's performance for detecting the existence of a turn-signal sound, for each vehicle, we construct positive and negative testing data. In particular, the positive testing data is a noisy turn-signal data collected from real-world driving, with the smartphone mounted close to the dashboard, i.e., mimicking real-world usage of smartphones while driving. The background noise includes mechanical noises and music sound while driving. We measured the noise level by putting a decibel meter on the dashboard. The overall noise volume (SPL) is approximately 65 dB, whereas the averaged turn-signal sound is around 46 dB. The negative testing data is a recording collected from a moving vehicle. To emulate the sound that will be recorded during steering maneuvers, we divided the sound into 3-second chunks (steering maneuvers for a



(a) Left/right turn test (b) Lane-change test (c) Results of steering detection

Figure 5.8: (a) and (b) show the test routes. We blurred the map and street names to protect the anonymity. (c) Shows the performance metric.

lane change take an average of 3.5 – 6 seconds [165]). In summary, each vehicle has an average of 15 positive and 10 negative sound snippets.

The experimental results are presented as a bar chart in Fig. 5.7 with the precision, recall and accuracy of each vehicle type. Our sound detection module is able to achieve average precision, recall, and accuracy of 0.94, 0.89, and 0.89, respectively. Note that Lincoln MKZ has the worst overall performance because of the low volume of its turn-signal sound.

## 5.4 Evaluation

### 5.4.1 Benchmark Test

We now benchmark `TurnsGuard`'s performance via real-world field testing. All field tests are done in controlled environments to obtain the ground truth. The driver used a Google Pixel XL (phone) and a Ford Explorer (car).

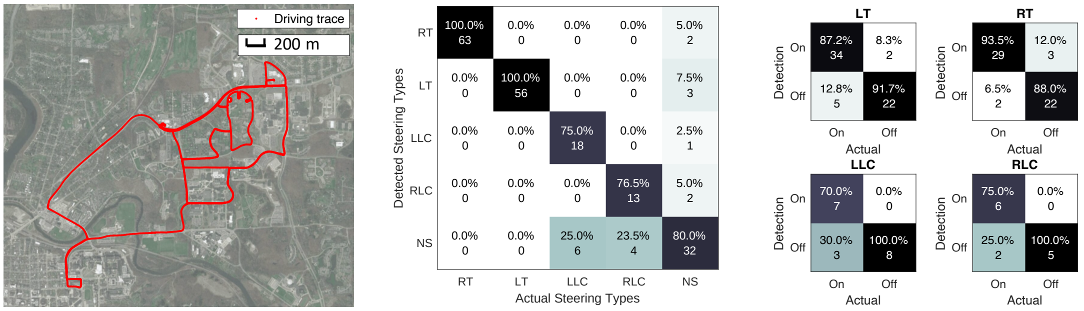
We conducted the left/right-turn experiments in the nights by driving on a pre-determined route in a large parking lot (as shown in Fig. 5.4.1). This experiment includes two paths: the driver first drove from the start to the end point via path 1, and then returned via path 2. The driver signaled 8 left turns (LTs) and 6 right turns (RTs) on path 1 and omitted the turn signal of 6 LTs and 6 RTs on path 2. The experiment consists of a total of 14 LTs and 12 RTs. The experimental runs for detecting left lane-changes (LLCs), right lane-changes (RLCs) and curved roads are shown in Fig. 5.4.1. `TurnsGuard` is tested/evaluated on a suburban road with two lanes in each direction, and moderate traffic. The participant drove on the route twice, each time making 5 lane-changes and encountering 3 curved road segments as shown in Fig. 5.4.1. The driver signaled all lane changes, abiding by the local traffic law and paying attention to safety.

We use a confusion matrix to describe TurnsGuard's steering detection performance in Fig. 5.4.1. We examined TurnsGuard's performance in detecting LT, RT, LLC, RLC, and maneuvers that are not an active steering maneuver (NS), where NS represents the detection of driving on curved roads. Each cell shows the number of occurrences and precision for each steering maneuver. In summary, TurnsGuard is shown to achieve the overall accuracy of 0.952. The turn detection module was able to detect all LTs and RTs correctly, which are consistent with the performance reported in [102]. The precision for detecting LLCs and RLCs with 0.8 precision. We noticed a mis-classification when the driver was making a lane-change in the middle of a curved road segment. This is because the trajectory of a lane-change on an L-shaped curved road can be very similar to driving on a straight road segment, thus leading to incorrect detection on a curved road. In both experiments, our turn-signal detection module is used to analyze the presence of a turn signal. In all 24 signaled steering maneuvers, TurnsGuard was able to detect 21 turn signals, whereas for 12 unsignaled steering maneuvers, TurnsGuard mis-detected 3 maneuvers as the signaled. This result indicates an overall TurnsGuard's accuracy of 0.83 in detecting turn-signal usage.

### 5.4.2 Large-scale Field Test

We further evaluate TurnsGuard's performance in a large-scale setting before starting a user study, because our user study is designed to represent participants' natural driving behavior, an intrinsic *unsupervised* nature that makes it infeasible to conduct comprehensive ground truth collection.

In this experiment, we have one participant (a 28-year-old male, with 5 years of



(a) Large-scale test trace (b) Steering detection (c) Sound detection

Figure 5.9: (a) shows the test routes. (b) and (c) show the performance metric.

driving experience) drove an SUV for 90.6km (1.35 hours) in the same region in a Saturday afternoon. The overall trace is shown in Fig. 5.9(a). During this experiment, the smartphone was placed in the cup-holder. To record the ground truth during the experiment, an annotator sat in the passenger seat to take a timestamped note of the participant's driving behavior. Specifically, each entry of the annotation is in the form:

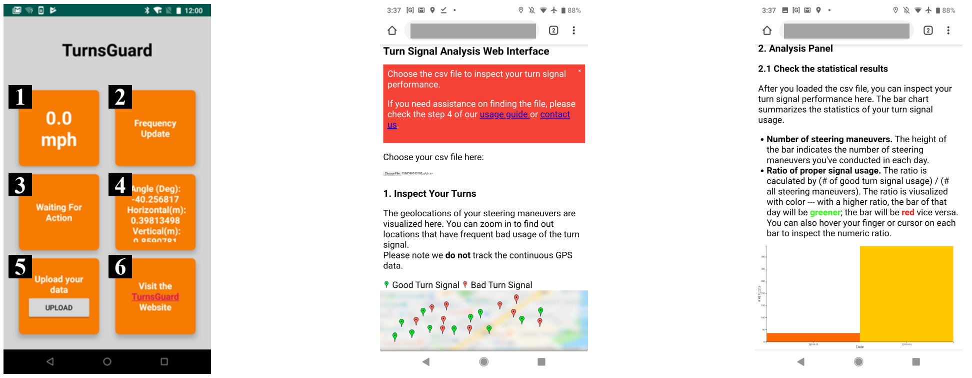
$$\{timestamp, steering\_type, signal\_usage.\} \tag{5.8}$$

To keep a reference of the ground truth, we used another phone (mounted on the windshield) for recording a video (including the sound) of the entire experiment. An OBD-II dongle is also used for recording the turn-signal usage. To mimic the real-world driving, the driver and the annotator were allowed to talk casually during the experiment. Specifically, there were regular conversations during 86% of the experiment, with an average volume of 66 dB. Our experiment resulted in 63 LTs, 56 RTs, 24 LLCs and 17 RLCs. `TurnsGuard`'s performance in the real-world setting is measured by:

- whether the steering detector can reliably capture the driver's steering maneuvers, and
- for all correctly detected steering maneuvers, whether the sound detection module can identify the existence of the clicking sound.

To investigate the first performance metric, the results of detecting steering maneuvers are presented as the confusion matrix in Fig. 5.9(b). Compared to the benchmark test, the large-scale evaluation includes more comprehensive tests of the steering detection module in differentiating steering maneuvers from driving on curvy roads, a challenging task associated with steering detection. To this end, our driving traces include several curvy road segments (as shown in Fig. 5.9(a)), in which `TurnsGuard` collects and uses the horizontal displacement for differentiating abnormal steering from normal steering. As a result, LT and RT achieve 100% detection rates, reproducing the performance of the small-scale benchmark test, with the overall `TurnsGuard`'s accuracy of 91% in detecting steering maneuvers. Note the mis-classifications of non-steering maneuvers (the rightmost column in Fig. 5.9(b)) — false positives that trigger sound detections even in the absence of steering maneuvers. This stems from diverse real-world road curvatures, which may yield vehicle movements similar to steering maneuvers. Since our test environment includes several curvy road segments (as shown in Fig. 5.9(a)), the steering detection can achieve better accuracy in regions with a grid street plan [70] where straight roads are predominant. Examples include New York, Chicago, and Barcelona to name a few.

For the second performance metric, we need to evaluate both positive and negative detection rates of our signal-detection pipeline. To collect the negative samples (i.e., steering without the turn-signal usage), the driver was instructed to omit the turn signal for some steering maneuvers. Since this behavior is dangerous and illegal in certain areas (e.g., State of California), all steering maneuvers without turn signals were made on low-traffic roads and in parking lots with no other vehicles in sight. In summary, 24 of 63 LTs, 25 of 56 RTs, 10 of 24 LLCs, and 7 of 17 RLCs are associated without turn-signal uses, whereas the remainders are with proper turn signals. In all successfully detected steering maneuvers, 39 LTs, 31 RTs, 10 LLCs, and 8 RLCs are associated with proper turn signals. The confusion matrices of turn-signal detection of the four steering maneuvers are shown in Fig. 5.9(c). `TurnsGuard`'s sound detection module shows a higher ratio of false negatives (based on the LLC and RLC results) than the false positive ratio. This is



Turn Signal Analysis Web Interface

Choose the csv file to inspect your turn signal performance.

If you need assistance on finding the file, please check the step 4 of our usage guide or contact us.

Choose your csv file here:

Choose File 1568599743180_old.csv

1. Inspect Your Turns

The geolocations of your steering maneuvers are visualized here. You can zoom in to find out locations that have frequent bad usage of the turn signal.
Please note we **do not** track the continuous GPS data.

Good Turn Signal Bad Turn Signal

2. Analysis Panel

2.1 Check the statistical results

After you loaded the csv file, you can inspect your turn signal performance here. The bar chart summarizes the statistics of your turn signal usage.

- **Number of steering maneuvers.** The height of the bar indicates the number of steering maneuvers you've conducted in each day.
- **Ratio of proper signal usage.** The ratio is caculated by (# of good turn signal usage) / (# all steering maneuvers). The ratio is viusalized with color --- with a higher ratio, the bar of that day will be greener; the bar will be red vice versa. You can also hover your finger or cursor on each bar to inspect the numeric ratio.

(a) App (b) Web interface (c) Progress tracking

Figure 5.10: Screen shots of TurnsGuard from a Google Pixel XL phone. In (a), block #1 shows the vehicle's current speed; #2 shows the debug message for the sound detection; #3 and #4 show the debug message for steering detection; #5 is the button for users to upload their collected data; #6 presents the link of our online analysis portal. We blurred the URL bar and the map view for the purpose of anonymity.

because of TurnsGuard's conservative (i.e., safety-first) design, lowering the false positive rate (FPR) by increasing the sensitivity of our sound detection module. Specifically, the sensitivity of the sound detection module can be increased by narrowing the search area as discussed in Sec. 5.3.4.4. The overall accuracy of sound detection for LT, RT, LLC, and RLC are 89%, 91%, 83%, and 85%, respectively. The lower success rates for lane changes are due to the smaller time required by lane-changes than left/right-turns.

## 5.5 Use TurnsGuard for Profiling Driving Maneuvers

To understand the user experience (especially from unbiased users) of TurnsGuard, we tested TurnsGuard in a user study.

### 5.5.1 Design of a User Study

We have designed a 5-phase empirical study (IRB approved, study number: HUM00167653) as described below.

#### 5.5.1.1 Phase 1. Recruitment of Participants

We launched the study of TurnsGuard on our university campus in a mid-sized town (approximately 75 $km^2$) in the U.S. This study chose university students since teenager drivers often lack driving experience and tend to be less attentive to driving. In fact, statistical data from auto insurance companies shows that drivers under age 26 are considered to be among the highest risk groups [60]. So, university students are likely to make un-signaled turns or lane-changes. It would also be interesting to study experienced drivers' usage of turn signals. For example, TurnsGuard can be used to compare the time needed for changing the driving habit for different age groups. We will leave this exploration as our future work.

We recruited 5 participants of ages 19 – 23. Since all participants drive their own cars for the experiment, we use 5 different cars in this study. Five different Android phones (two Nexus 5X, one Samsung S10, one Samsung S9+, and one Pixel XL) are used in our study. All participants are asked to consent to our study before starting their experimental runs.

#### 5.5.1.2 Phase 2. Building a Sound Profile

The participants are asked to collect a 10-second recording of their vehicle's clicking sound for customizing the front-end app for their car. The detailed steps are presented in Sec. 5.3.1.

#### 5.5.1.3 Phase 3. Collection of Natural Driving Data

Our user study focuses on capturing the driver's natural turn-signal usage. We reward every participant with financial compensation ($20) if s/he can finish the study. All participants are informed that the study is to understand turn signal usage pattern with smartphones. To mitigate psychological biases, we stressed that finishing the study is the only requirement for receiving the reward. All participants are asked to drive naturally with their smartphone should be placed at a stable position, e.g., a cupholder or windshield mount. No restrictions are imposed on talking during this study. Note that our restriction do not require anything that may endanger drivers — both comply to the safe driving guidelines.

#### 5.5.1.4 Phase 4. Data Contextualization

For contextualizing the detection results, we design and deploy a web interface for visualizing the user's historical data.

The web interface for this study includes two components: (1) a Google map with pins for highlighting locations where the driver failed to use the signal for steerings. As shown in Fig. 5.10 (b), we use red pins for showing *bad* turn signal usage. (2) a colored bar chart to show the driver's statistical results. As shown in Fig. 5.10(c), the height of the bar represents the number of steering maneuvers made each day. The color of each bar is mapped from the ratio of the number of good turn signals to all steering maneuvers on that day. As a result, if the driver has a higher ratio on a day, the bar of that day will be greener; otherwise, the bar will be red. Drivers can also hover their fingers or cursor over each bar to get the numerical details. This allow the driver to understand his/her behavioral change through time.

#### 5.5.1.5 Phase 5. User Survey of `TurnsGuard`

We designed five question in our post-experiment survey. First, how frequent you drive and use our app? Second, is `TurnsGuard` easy to use (on a scale of 1 to 5)? Third, how useful do you think `TurnsGuard` can help drivers enhance their attentiveness to turn signals? Fourth, how do you think (on a scale of 1 to 5) the contextualization channel (web interface) can improve the driver's self-awareness of turn-signal usage? Fifth, please share any of your feedback with us.

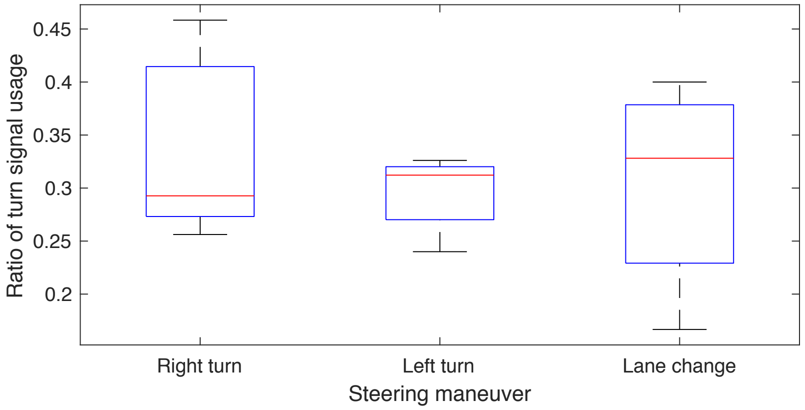


Figure 5.11: Distribution of users' turn signal usage pattern.

### 5.5.2 Results and Insights

According to the answers of the first question, three of five participants have used TurnsGuard for more than five days, whereas the other two participants have used it for 2–4 days. All participants have, on average, more than 20 minuets driving time per day. For the second survey question, four out of five participants gave 5 (very easy to use), three participants explicitly praised (answers for the fifth question) TurnsGuard's *privacy preservation* design after they learned the technical detail of our sound detection process — it does not collect sound recordings but record only the data related to steering maneuvers. For the third question, two of the five participants think TurnsGuard helps them enhance their turn-signal usage by giving a score higher than 3, whereas two participants gave a score of 2. One participant gave a score of 3 for this question. We will show the result of the fourth question in Sec. 5.5.2.2.

#### 5.5.2.1 Turn-Signal Usage Pattern

The data collected from our user study includes 572 left turns, 669 right turns, and 118 lane-changes. We first look at the the turn signal usage pattern of all participants by analyzing the ratio of correct turn-signal usage (i.e., a turn signal is detected for the corresponding steering maneuver) of different steering maneuvers. As shown in Fig. 5.11,

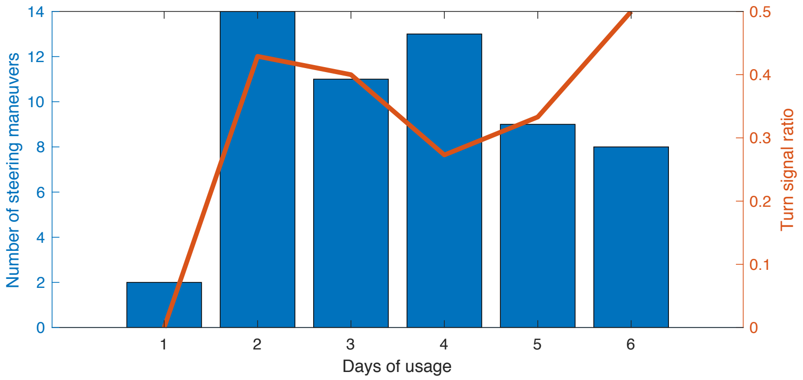


Figure 5.12: Statistical change of the turn-signal usage over time.

the turn signal use ratio of left turns (median = 0.31) shows higher median and lower variance than that of right turns (median = 0.29). We conjecture this is because left turns are *risker* than right turns, and hence drivers pay more attention when they make left turns. Note that the turn signal ratio of lane changes shows the highest median (0.33) and a high variance, because different drivers may have very different patterns/habits of turn signal usage even though drivers are aware of the importance of turn-signal usage during a lane-change.

#### 5.5.2.2 Users' Awareness of Turn Signal Usage

According to participants' answer of the fourth question, three of all five participants agreed (by giving a score higher than 3) that reviewing the historical data via the colored bar chart is straightforward and can effectively enhance their self-awareness of turn-signal usage, while the other two participants gave a score of 2. For the data analytics, after a user examined his/her performance at the end of day 1 and 3, we can observe (from Fig. 5.12) improvements of the turn-signal usage ratio after these two days. Although the improvement is subtle and need longer term experiment to support, it hints that our intervention (e.g., web interface) may help improve the user's attentiveness.

Note that, `TurnsGuard` also allows the app developer to design a stronger intervention arm for improving the driver's attentiveness. For example, a *real-time* alert (e.g., achieved by using the pop-up notification on Android and iOS) can be implemented for reminding the user of the neglect of the turn signal. The real-time alert functionality can be used in several mobility apps, including driving coaching and parental control apps, to name just a few.

### 5.5.3 Limitations

Based on the user study and participants' feedback, there are following limitations of `TurnsGuard`.

#### 5.5.3.1 Technical Limitations

One participant complained `TurnsGuard` cannot filter out the turning maneuvers at a parking lot. This limitation could result in many "bad turns" at a parking lot. To solve this problem, we plan to filter the parking lot steering maneuvers by using contextual information, such as smartphone's GPS speed.

#### 5.5.3.2 Incentivizing Drivers

We currently use financial incentives to attract participants to use `TurnsGuard` and its online interfaces. This approach will not scale to a large number of users. An alternative method for incentivizing users is collaboration with transportation stakeholders who are interested in the drivers' attentiveness and/or safety. For example, insurance programs like UBI reward safe driving with premium discounts.

## 5.6 Discussion

### 5.6.1 Beyond Detection of Turn Signals

Knowing whether or not the driver is using the turn signal properly is essential for various applications. Discussed below are the potential safety-critical information and new research directions that can be enabled by `TurnsGuard`.

#### 5.6.1.1 1. Analyzing Road Safety with `TurnsGuard`

As `TurnsGuard`'s users-base grows and detection data accumulates, we can also gather data on intersections in a specific area. The collected data by `TurnsGuard` can be associate with other traffic information (e.g., local accident report) to evaluate the traffic safety of that area. Moreover, an intersection with high turn signal usage can be regarded as a safe intersection, while a turn with low turn-signal usage should be flagged and warned as unsafe. Meanwhile, outlier turn signal uses can be detected from this information. That is, a driver making a bad (unsignaled) turn at a high-turn-signal-usage road segment may indicate dangerous driving, and should be alerted and even penalized.

Furthermore, detection of a large number of outlier turn signal misuses can be an indication of changes in traffic condition. These abnormalities in data can serve as an additional feature in traffic monitoring. An area with an abrupt eruption of outlier turn signal uses should be investigated closely and monitored by the local transportation department and/or traffic enforcers.

#### 5.6.1.2 2. Detection of Consecutive Misuses of Turn Signals.

A driver makes consecutive turns with/without using turn signals may indicate changes in his/her driving behavior and condition. Specifically, as a person drives, `TurnsGuard` accumulates the data of his/her turns it has detected. We can, therefore, analyze this accumulated data, which may indicate changes in the person's driving behavior and condition. A driver with a bad turns record now making good turns consecutively may sense improvements in his/her driving. A driver with a good turns records making consecutive bad turns may indicate his/her careless driving due to something that bothers him/her, or due to a sudden change in the road/traffic condition.

We can further use this extracted information to improve the effectiveness of our notification system. An improvement in driving behavior should be rewarded with incentivizing texts, while a worsening driving behavior should be flagged, and prompt notifications such as ”be wary of careless driving” and ”remember to use turn signals in all road/traffic conditions”.

#### 5.6.1.3 3. Adaptive/Customized Path Planning with TurnsGuard

Most current research on vehicle navigation focuses on designing increasingly optimal and efficient algorithms to find the least cost path. The existing cost metrics are mostly temporal (travel time) or spatial (travel distance) [52]. The lack of driver-specific metrics makes it difficult to customize current navigation algorithms for individual drivers. TurnsGuard provides a new cost (behavioral) metric, and enables adaptive and/or customizable navigation.

The driving data gathered from TurnsGuard models the likelihood of correct turn signal usage, which reflects driving behavior. Moreover, the data can model turn signal usage at intersections on a specific path. This modeling provides a behavioral metric that navigation algorithms can optimize a navigation path that matches the modeled driving behavior. This makes the navigation path not only optimal in spatial and temporal sense, but also adaptive to the driver.

## 5.7 Conclusion

We have presented TurnsGuard, the first profiling tool for monitoring and analyzing the driver’s turn-signal usage by only using commodity smartphones. TurnsGuard is shown to be able to profile (1) turn-signal usage during steering maneuvers, and (2) the dynamics of the driver’s steering maneuver. By exploiting these capabilities of TurnsGuard on the user’s smartphone, our experimental tests and on-road user study demonstrate TurnsGuard’s capability of studying drivers’ turn-signal usage at a large scale.

CHAPTER VI

# Conclusion and Future Directions

## 6.1 Conclusion

This thesis has demonstrated the feasibility of using ubiquitous sensing and machine learning to enable a practical feedback loop (as discussed in Chapter I) to benefit the mobility ecosystem. It has detailed four closely-knit systems — `V-Sense` (Chapter II), `Dri-Fi` (Chapter III), `TurnsMap` (Chapter IV), and `TurnsGuard` (Chapter V)that present accessible, reliable, and usable systems to facilitate seamless HMI.

### 6.1.1 Accessibility

These four systems can be implemented on off-the-shelf mobile platforms, which greatly enhances their accessibility of the proposed systems. Compared to other sensory data (e.g., camera and sound), IMU sensors are universally supported by the vast majority of mobile devices. Due to the essential roles of IMU sensors in capturing device dynamics, including quickly emerging augmented reality (AR) applications, these sensors are unlikely to be replaced in the future. Moreover, the generation of time-series data incurs low energy and computational overhead even for continuous sensing tasks (i.e., always-on sensors) as evidenced by our evaluation results.

### 6.1.2 Reliability

As discussed in Chapter I, raw IMU sensor data produced by mobile devices may be noisy given the devices changing posture (e.g., in a drivers pocket vs. a cupholder) and low sensor fidelity. Overcoming these obstacles is pivotal to ensuring the reliability of our four proposed systems.

**Data calibration.** Data calibration, namely coordinate alignment (see Chapter II), is important in addressing the varying-posture problem. All of our systems applied coordinate alignment to users devices (i.e., in the front end) to transform a changing coordinate system to a fixed earth coordinate system. By doing so, IMU sensor data, such as from gyroscopes, can be compressed into a one-dimensional data stream from the original three-dimensional vector (i.e., data that reflect pitch, yaw, and roll axes). Coordinate alignment can also reduce the data dimension to achieve more efficient data collection and analysis.

**Data preprocessing.** As shown in these four systems, basic data preprocessing steps (e.g., use of a low-pass filter) can alleviate noisy sensor readings, including spikes following from system glitches. The specific requirements of different applications should also be considered during data preprocessing. `V-Sense` and `TurnsGuard` preprocess IMU data locally on the users device given the real-time requirements of their design goals; for example, `TurnsGuard` must alert the driver on unsignaled steering maneuvers. By contrast, data processing for `Dri-Fi` and `TurnsMap` can be implemented in the cloud because their data need not be in real-time. As indicated via our evaluations (Chapter III), simplifying computational tasks on users devices can mitigate the associated energy overhead.

### 6.1.3 Usability

Usability is an essential feature of our works, as all four systems were motivated, designed, and evaluated based on practical problems.

**Motivation.** `V-Sense` is intended to mitigate cameras limitations in detecting vehicle steering. `Dri-Fi` was motivated by the complicated nature of driver identification, which poses a major obstacle to fast-emerging UBI and ride-sharing businesses. TurnsMap was inspired by the riskiness of unprotected left turns at intersections, which many drivers reported as a complaint in our user study. With TurnsGuard, we have developed the first scalable tool to identify the dangerous driving habit of unsignaled steering.

**System Design.** All our system designs were inspired by real-world observations. For instance, the strong feature (i.e., the concave shape of a smartphones gyroscope reading) resulting from interruptions during unprotected left turns (Chapter IV) was discovered using driving data collected by our V-Sense app. Dri-Fi discovers driver's steering maneuver as the *behavioral-rich* maneuver for characterizing drivers

**Evaluation.** To ensure the usability of our systems in the real-world, we have evaluated these four systems with natural driving data gathered from drivers mobile devices while driving. For example, Dri-Fi uses natural steering data to characterize each drivers driving pattern. We also split real-world data into control and experimental groups to assess Dri-Fis performance. In TurnsMap, we split the accumulated driving data at intersections into training and testing datasets for building machine learning model and evaluating our system, respectively.

## 6.2 Future Directions

The four systems presented herein suggest several exciting research directions. First, with more sophisticated ubiquitous sensors, one could potentially analyze more sophisticated data. Second, privacy-preserving computing paradigms, including data collection and analytics, could prevent data misuse and better protect users privacy.

### 6.2.1 Exploring New Sensor Types

Within the rapidly evolving computing paradigm of advanced mobility technologies, new accessible sensing modules can generate context-rich data that can be used to infer more implicit information. This thesis research can be extended by investigating other novel sensor types to enhance mobility systems.

As cars become *smarter* and more connected, modern vehicles are equipped with a growing number of electronic control units, many of which can generate rich data reflecting vehicle dynamics, engine health, and other contextual information. As noted in Chapter III, the CAN data format is proprietary to OEMs, and reverse engineering may involve prohibitive time and effort. To address this challenge, LibreCAN [155] was recently developed to automatically translate most CAN messages. This automated translator of proprietary data can be used to break the barrier of remote in-vehicle data access and render car-based data flow transparent to developers. As a short-term task, I would like to evaluate the feasibility of using translated in-vehicle data to develop new applications based on reverse-engineered vehicular data.

However, harvesting data from diverse mobility systems remains in its infancy; in the long term, I plan to continue pursuing this research direction by exploring (a) additional sensor types (e.g., sensors in roadside infrastructure) and (b) novel applications that can be enabled by increasingly advanced ubiquitous sensors.

### 6.2.2 Usable Privacy Model

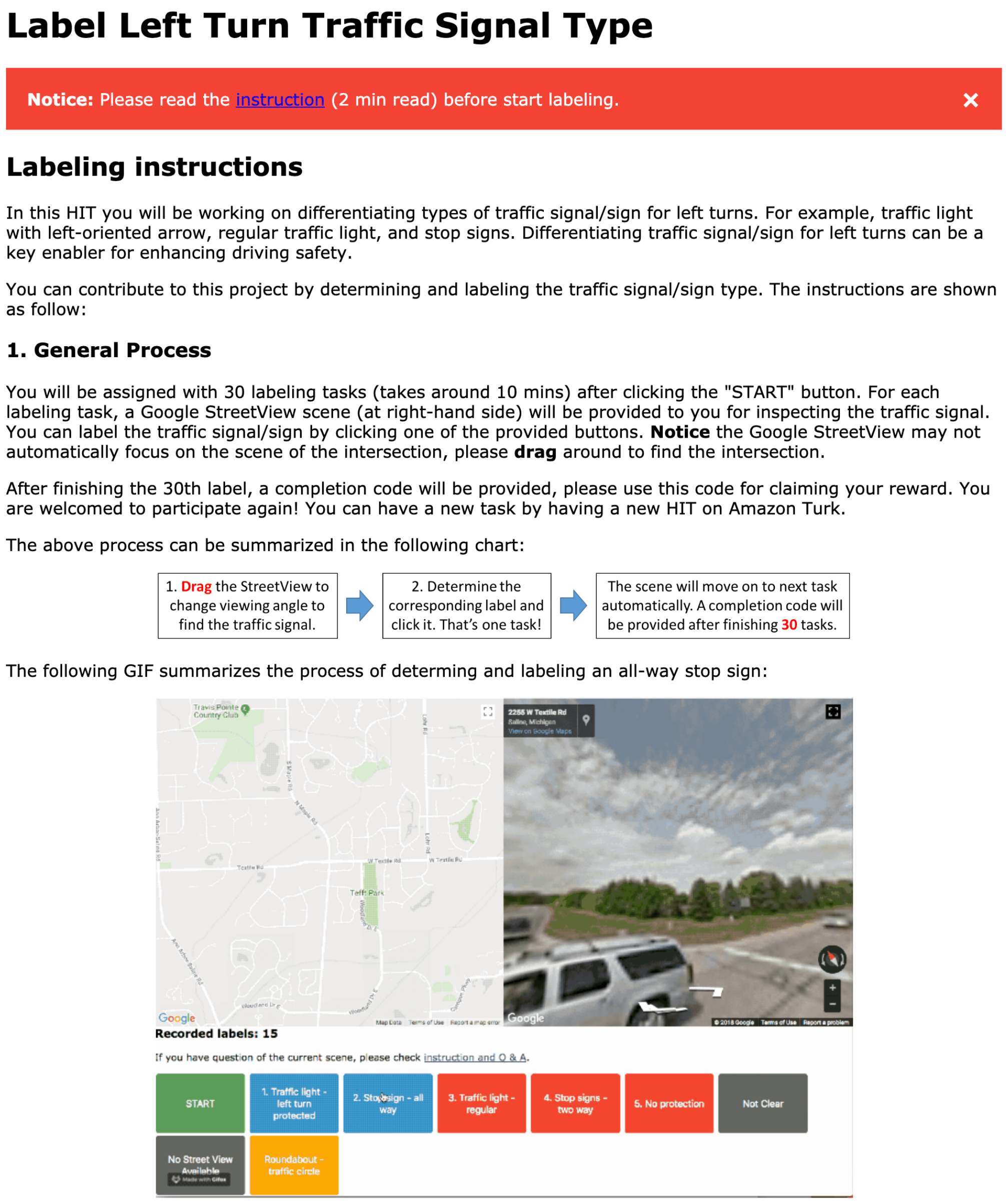
Another dimension to enhance user privacy will involve studying potential side-channel information leakage via mobile sensing. Specifically, I plan to investigate what side-channel information may be leaked by malicious data controllers and/or processors (as defined in General Data Protection Regulation [80] Article 4). For example, peoples chosen vehicle types (e.g., a Toyota Prius hybrid car vs. a Ford F-150 pickup truck) contain information-rich data that may carry implications for various forms of privacy-related information, such as driver demographics. Existing studies [120, 124], market study [21], and surveys [11, 16, 50] have also indicated that peoples vehicle choices could reflect their political stance (e.g., conservative or liberal).

In the long term, I plan to study how to protect users privacy while preserving the usability of novel sensing technologies. This is a challenging task because enhanced sensing technology could be intrusive, whereas privacy-preserving sensing technologies may undermine its usability. For example, `Dri-Fi` and `TurnsMap` have demonstrated usability in real-world applications. Although they provide a pathway to various beneficial applications, we must not overlook how their capabilities (i.e., driver identification and location tracking) may raise users privacy concerns. To address these concerns, I plan to investigate a privacy model to protect users privacy by adding noise systematically, such as use of a differential privacy model [112].

Moving forward, I would like to explore how the seamless HMI can help facilitate the coexistence between self-driving and human-driven cars. For example, it took humans over a century to upgrade the mobility ecosystem from horse carriages to the modern motorized vehicles. This historical transition was a time-consuming and tough process that requires decades of experiments and adjustments of new technologies/infrastructures, including vehicle components (e.g., ABS), traffic regulations (e.g., stop signs and traffic lights), road infrastructure (e.g., roundabout, pedestrians crossing bridge). Seamless HMI can enable an accessible and scalable data analytics scheme of different mobility systems, i.e., the feedback loop discussed in Chapter I. This might be able to expedite the transition process of inventing and/or refining mobility technologies, thus making the future transportation safer and more efficient.

# APPENDICES

## .1 The Tutorial Page of TurnsMap



### Label Left Turn Traffic Signal Type

**Notice:** Please read the instruction (2 min read) before start labeling. ✕

### Labeling instructions

In this HIT you will be working on differentiating types of traffic signal/sign for left turns. For example, traffic light with left-oriented arrow, regular traffic light, and stop signs. Differentiating traffic signal/sign for left turns can be a key enabler for enhancing driving safety.

You can contribute to this project by determining and labeling the traffic signal/sign type. The instructions are shown as follow:

**1. General Process**

You will be assigned with 30 labeling tasks (takes around 10 mins) after clicking the "START" button. For each labeling task, a Google StreetView scene (at right-hand side) will be provided to you for inspecting the traffic signal. You can label the traffic signal/sign by clicking one of the provided buttons. **Notice** the Google StreetView may not automatically focus on the scene of the intersection, please **drag** around to find the intersection.

After finishing the 30th label, a completion code will be provided, please use this code for claiming your reward. You are welcomed to participate again! You can have a new task by having a new HIT on Amazon Turk.

The above process can be summarized in the following chart:

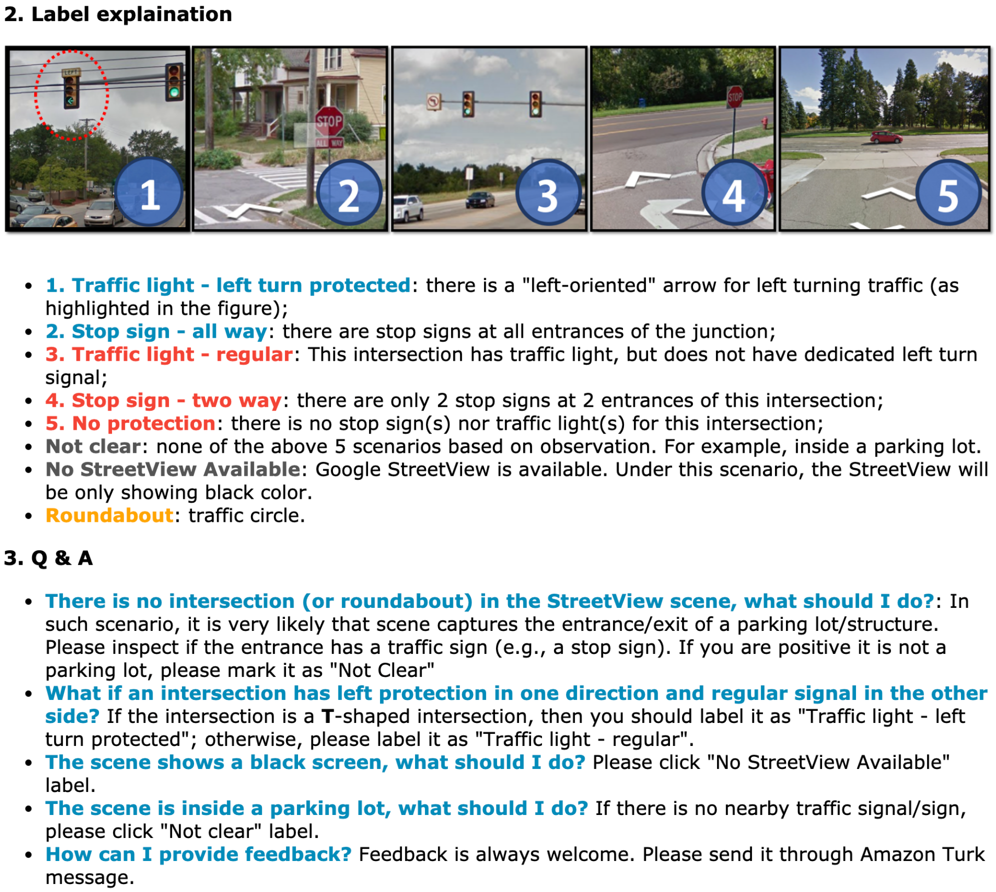
1. **Drag** the StreetView to change viewing angle to find the traffic signal. →
2. Determine the corresponding label and click it. That's one task! →
The scene will move on to next task automatically. A completion code will be provided after finishing **30** tasks.

The following GIF summarizes the process of determing and labeling an all-way stop sign:

**Recorded labels: 15**

If you have question of the current scene, please check instruction and Q & A.

(a) Step 1



**2. Label explaination**

- **1. Traffic light - left turn protected**: there is a "left-oriented" arrow for left turning traffic (as highlighted in the figure);
- **2. Stop sign - all way**: there are stop signs at all entrances of the junction;
- **3. Traffic light - regular**: This intersection has traffic light, but does not have dedicated left turn signal;
- **4. Stop sign - two way**: there are only 2 stop signs at 2 entrances of this intersection;
- **5. No protection**: there is no stop sign(s) nor traffic light(s) for this intersection;
- **Not clear**: none of the above 5 scenarios based on observation. For example, inside a parking lot.
- **No StreetView Available**: Google StreetView is available. Under this scenario, the StreetView will be only showing black color.
- **Roundabout**: traffic circle.

**3. Q & A**

- **There is no intersection (or roundabout) in the StreetView scene, what should I do?**: In such scenario, it is very likely that scene captures the entrance/exit of a parking lot/structure. Please inspect if the entrance has a traffic sign (e.g., a stop sign). If you are positive it is not a parking lot, please mark it as "Not Clear"
- **What if an intersection has left protection in one direction and regular signal in the other side?** If the intersection is a **T**-shaped intersection, then you should label it as "Traffic light - left turn protected"; otherwise, please label it as "Traffic light - regular".
- **The scene shows a black screen, what should I do?** Please click "No StreetView Available" label.
- **The scene is inside a parking lot, what should I do?** If there is no nearby traffic signal/sign, please click "Not clear" label.
- **How can I provide feedback?** Feedback is always welcome. Please send it through Amazon Turk message.

(b) Step 2 & 3

Figure .1: The tutorial for annotators to read before start labeling.

# BIBLIOGRAPHY

# BIBLIOGRAPHY

[1] Accelerating the pace of learning. https://medium.com/waymo/accelerating-the-pace-of-learning-36f6bc2ee1d5.

[2] Amazon mechanical turk requester ui guide. https://docs.aws.amazon.com/AWSMechTurk/latest/RequesterUI/amt-ui.pdf.

[3] Android fragmentation (august 2015). https://opensignal.com/reports/2015/08/android-fragmentation/.

[4] Anti-lock braking system. https://en.wikipedia.org/wiki/Anti-lock_braking_system#cite_note-absfaq-25.

[5] Augmented driving. https://itunes.apple.com/us/app/augmented-driving/id366841514?mt=8.

[6] Blacksensor on google play. https://play.google.com/store/apps/details?id=com.chahoo.bsdrive&hl=en.

[7] Bosch BM160 IMU chip data sheet. https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMI160-DS000-07.pdf.

[8] Cellular connection of apple watch 3. https://www.apple.com/watch/cellular/.

[9] Coverage of google streetview. https://en.wikipedia.org/wiki/Coverage_of_Google_Street_View.

[10] Device for indicating the intended movements of vehicles.

[11] Do You Drive a Liberal Car or Conservative Car? https://baristanet.com/2012/08/do-you-drive-a-liberal-car-or-conservative-car/.

[12] Drivea app. http://www.drivea.info/.

[13] Fatality analysis reporting system (fars) encyclopedia. http://www-fars.nhtsa.dot.gov/Main/index.aspx.

[14] The first look inside zooxs mysterious robo-taxi. https://www.bloomberg.com/news/articles/2017-11-29/the-first-look-inside-zoox-s-mysterious-robo-taxi.

[15] Fox business: Half of drivers don't use turn signals. https://www.foxbusiness.com/features/half-of-drivers-dont-use-turn-signals.

[16] Gallup Poll Analysis: Political Correlates of Car Choice. http://news.gallup.com/poll/23230/gallup-poll-analysis-political-correlates-car-choice.aspx.

[17] Google map enable lane guidance. https://support.google.com/gmm/answer/3273406?hl=en.

[18] Google's waze is helping drivers avoid left-hand turns. http://fortune.com/2016/06/18/google-waze-difficult-intersection/.

[19] Haversine formula. https://en.wikipedia.org/wiki/Haversine_formula.

[20] Here hd map. https://www.here.com/en.

[21] Here's What Really 'Drives' Democrats And Republicans. https://www.forbes.com/sites/jimgorzelany/2016/11/02/heres-what-really-drives-democrats-and-republicans/#19a7e85e22b8.

[22] Honda, advanced driver-assistive system. http://world.honda.com/news/2014/4141024Honda-SENSING-Driver-Assistive-System/.

[23] How does google maps calculate your eta? https://www.forbes.com/sites/quora/2013/07/31/how-does-google-maps-calculate-your-eta/#e1dc5a3466e2.

[24] How google tracks traffic. https://web.archive.org/web/20140222173908/http://www.theconnectivist.com/2013/07/how-google-tracks-traffic/.

[25] Intersection design. http://www.deldot.gov/information/pubs_forms/manuals/road_design/pdf/revisions062811/07_Intersections.pdf?100411.

[26] Intersections and right of way. https://www.dmv.org/how-to-guides/intersections-and-right-of-way.php.

[27] Introduction to the controller area network (can). http://www.ti.com/lit/an/sloa101b/sloa101b.pdf.

[28] ionroad app. http://www.ionroad.com/.

[29] k nearest neighbor dynamic time warping. http://www.cs.unm.edu/~mueen/DTW.pdf.

[30] Manual on uniform traffic control devices. https://mutcd.fhwa.dot.gov/pdfs/2009r1r2/mutcd2009r1r2edition.pdf.

[31] Mobileye. http://www.mobileye.com/.

[32] Number of vehicles in operation in the united states between 1st quarter 2017 and 1st quarter 2019 (in millions). https://www.statista.com/statistics/859950/vehicles-in-operation-by-quarter-united-states/.

[33] Open data portal for traffic light at intersection. http://gisdata-arlgis.opendata.arcgis.com/datasets/af497e2747104622ac74f4457b3fb73f_2.

[34] Over 1,400 self-driving vehicles are now in testing by 80+ companies across the us. https://techcrunch.com/2019/06/11/over-1400-self-driving-vehicles-are-now-in-testing-by-80-companies-across-the-u-s/.

[35] Pew research: Smartphone ownership across the world. http://www.pewglobal.org/2016/02/22/smartphone-ownership-and-internet-usage-continues-to-climb-in-emerging-economies/.

[36] Proposed features/turn signals. http://wiki.openstreetmap.org/wiki/Proposed_features/turn_signals#Traffic_signal_only_for_traffic_that_turns_left.

[37] Road edge and barrier detection with steer assist. https://www.youtube.com/watch?v=xrLVaWnJmMI.

[38] Safety issues of left turns at intersections. https://www.washingtonpost.com/news/innovations/wp/2014/04/09/the-case-for-almost-never-turning-left-while-driving/.

[39] Sleepy behind the wheel? some cars can tell. https://www.nytimes.com/2017/03/16/automobiles/wheels/drowsy-driving-technology.html.

[40] Statista: Number of connected iot devices across the world. https://www.statista.com/statistics/789615/worldwide-connected-iot-devices-by-type/.

[41] Steering patterns as drowsy indicator. http://www.sae.org/events/gim/presentations/2012/sgambati.pdf.

[42] Tomtom hd map. https://www.tomtom.com/en_us/.

[43] Traffic signal data on data.gov. https://catalog.data.gov/dataset?q=traffic+lights&sort=none.

[44] Turn left at a traffic light safely. http://drivinginstructorblog.com/turn-left-traffic-lights/.

[45] Turning radius and intersection size. http://articles.latimes.com/2005/apr/20/autos/hy-wheel20.

[46] Volvo xc90. http://www.volvocars.com/us/cars/new-models/all-new-xc90.

[47] Waymo tests its self-driving cars in my town. here are the odd things i've seen. https://www.forbes.com/sites/rrapier/2017/11/12/waymo-tests-its-self-driving-cars-in-my-town-here-are-the-odd-things-ive-seen/#57a5eb724e4a.

[48] Waymos big ambitions slowed by tech trouble. https://www.theinformation.com/articles/waymos-big-ambitions-slowed-by-tech-trouble.

[49] Why ups trucks never turn left. https://www.cnn.com/2017/02/16/world/ups-trucks-no-left-turns/index.html.

[50] Your cars: politics on wheels. https://www.nytimes.com/2005/04/01/automobiles/your-car-politics-on-wheels.html.

[51] *Turn Signal Flashers*, jul 1999.

[52] A real-time vehicle navigation algorithm in sensor network environments. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6220254, 2012.

[53] Sound pressure level. https://en.wikipedia.org/wiki/Sound_pressure, 2012.

[54] The OpenXC Platform. http://openxcplatform.com/, 2012.

[55] *Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems*, jan 2014.

[56] Why dont we use turn signals? https://www.ajc.com/news/why-don-use-turn-signals/NGYxrMMl05uPJlJrxEiWZP/, 2014.

[57] Allstate mulls selling driver data. https://www.bloomberg.com/news/articles/2015-05-28/allstate-seeks-to-follow-google-as-ceo-mulls-selling-driver-data, 2015.

[58] CAN-bus can be encrypted. http://www.eetimes.com/document.asp?doc_id=1328081, 2015.

[59] Car Hacking. https://www.wired.com/2015/03/60-gadget-thatll-make-car-hacking-easier-ever/, 2015.

[60] Car Insurance for Teen After Accident. https://www.autoinsurance.org/auto-insurance-for-teens-after-an-accident/#Factors_Influencing_Rate_Hikes, 2016.

[61] Progressive's note on excluded drivers. https://www.progressive.com/glossary/, 2016.

[62] Stolen Uber accounts worth more than stolen credit cards. https://www.cnbc.com/2016/01/19/stolen-uber-accounts-worth-more-than-stolen-credit-cards.html, 2016.

[63] TrueMotion – Motioning in a New Wave of Auto Insurance. https://medium.com/@TylerCrown/truemotion-motioning-in-a-new-wave-of-auto-insurance-44ef48e362cb, 2016.

[64] Turn signal neglect is a leading cause of motor vehicle accidents in the u.s. https://www.shanestafford.com/turn-signal-neglect-causes-motor-vehicle-accidents/, 2016.

[65] Android foreground service. https://developer.android.com/guide/components/services, 2017.

[66] Driver Accounts are being Hacked. https://ridesharecentral.com/check-uber-driver-account-hacked-uber-data-breach-update, 2017.

[67] Fence api overview. https://developers.google.com/awareness/android-api/fence-api-overview, 2017.

[68] Flaw in CAN bus. https://www.wired.com/story/car-hack-shut-down-safety-features/, 2017.

[69] German car industry plans to close OBD interface. http://www.eenewsautomotive.com/news/german-car-industry-plans-close-obd-interface, 2017.

[70] Grid plan. https://en.wikipedia.org/wiki/Grid_plan#United_States, 2017.

[71] Nasdaq – Introducting Usage-based Insurance. http://www.nasdaq.com/article/driving-the-future-of-blockchains-part-four-introducing-usage-based-insurance-cm818062, 2017.

[72] Pokemon Go Speed Restrictions. https://bgr.com/2017/03/20/pokemon-go-speed-limit-bypass-trick/, 2017.

[73] Turn signals - common sense and common courtesy. https://www.transportation.gov/connections/turn-signals-common-sense-common-courtesy, 2017.

[74] Uber forbid driver account sharing. https://help.uber.com/h/1d93388d-cf19-408f-9c41-743dbdd34d44/, 2017.

[75] Usage based insurance. https://en.wikipedia.org/wiki/Usage-based_insurance, 2017.

[76] Using oauth 2.0 to access google apis. https://developers.google.com/identity/protocols/oauth2, 2017.

[77] Discussion on Reddit: My postmates driver is different. https://www.reddit.com/r/postmates/comments/6x23no/driver_was_someone_else/, 2018.

[78] Mckinsey: telematics poised for strong global growth. https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/telematics-poised-for-strong-global-growth, 2018.

[79] New Scam Puts Unchecked Rideshare Drivers Behind the Wheel. https://www.nbcbayarea.com/news/local/new-scam-puts-unchecked-rideshare-drivers-behind-the-wheel/207917/, 2018.

[80] The EU General Data Protection Regulation (GDPR). https://www.eugdpr.org/, 2018.

[81] Amazon is now making its delivery drivers take selfies. https://www.theverge.com/2019/4/19/18507789/amazon-delivery-drivers-selfies-facial-recognition-fraud-protection-flex-app, 2019.

[82] CNN investigation: 103 Uber drivers accused of sexual assault or abuse. https://money.cnn.com/2018/04/30/technology/uber-driver-sexual-assault/index.html, 2019.

[83] Getting to appreciate the unloved turn signal. https://www.nytimes.com/2019/06/20/smarter-living/use-your-turn-signal.html, 2019.

[84] Java PMML API. https://github.com/jpmml, 2019.

[85] Law Enforcement Struggles to Catch Fraudulent Uber and Lyft Drivers. https://www.nbcbayarea.com/news/local/bay-legal-uberlyft-scam-law-enforcement-struggles-to-catch-fraudulent-rideshare-drivers/6836/, 2019.

[86] They Thought It Was Their Uber. But the Driver Was a Predator. https://www.nytimes.com/2019/04/04/us/fake-uber-driver-assaults.html#commentsContainer, 2019.

[87] USAGE-BASED INSURANCE AND TELEMATICS. https://www.naic.org/cipr_topics/topic_usage_based_insurance.htm, 2019.

[88] About Face ID advanced technology. https://support.apple.com/en-us/HT208108, 2020.

[89] Balanced Accuracy. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html, 2020.

[90] Fake Grubhub Driver Charged With Hitting Restaurant Worker With His Car, Driving Away. https://blockclubchicago.org/2020/05/19/grubhub-driver-charged-with-hitting-restaurant-worker-with-his-car-driving-away-police-say/, 2020.

[91] Uber safety tips. `https://www.uber.com/us/en/ride/safety/tips/`, 2020.

[92] Uber to use its selfie tech to verify drivers are wearing masks. `https://techcrunch.com/2020/05/07/uber-may-use-its-selfie-tech-to-verify-drivers-are-wearing-masks/#:~:text=The%20driver%20selfie%20technology%2C%20officially,being%20allowed%20to%20accept%20fares.`, 2020.

[93] T. Abdelzaher, Y. Diao, J. L. Hellerstein, C. Lu, and X. Zhu. *Introduction to Control Theory And Its Application to Computing Systems*, pages 185–215. Springer US, Boston, MA, 2008.

[94] H. Aly, A. Basalamah, and M. Youssef. Map++: A crowd-sensing system for automatic map semantics identification. In *2014 Eleventh Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 546–554, June 2014.

[95] M. Aly. Real time detection of lane markers in urban streets. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 7–12, June 2008.

[96] K. J. Åström. Challenges in control education. In *7th IFAC Symposium on Advances in control Education, Madrid, Spain*, 2006.

[97] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, May 2017.

[98] C. Bo, X.-Y. Li, T. Jung, X. Mao, Y. Tao, and L. Yao. Smartloc: Push the limit of the inertial sensor based metropolitan localization using smartphone. In *Proceedings of the 19th Annual International Conference on Mobile Computing and Networking*, MobiCom'13, 2013.

[99] A. Bouhoute, R. Oucheikh, K. Boubouh, and I. Berrada. Advanced driving behavior analytics for an improved safety assessment and driver fingerprinting. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–14, 2018.

[100] Lane change at intersection in california. `http://articles.latimes.com/2005/apr/20/autos/hy-wheel20`.

[101] F. Caron, E. Duflos, D. Pomorski, and P. Vanheeghe. Gps/imu data fusion using multisensor kalman filtering: introduction of contextual aspects. *Information Fusion*, 7(2):221 – 230, 2006.

[102] D. Chen, K.-T. Cho, S. Han, Z. Jin, and K. G. Shin. Invisible sensing of vehicle steering with smartphones. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '15, pages 1–13. ACM, 2015.

[103] D. Chen, K.-T. Cho, and K. G. Shin. Mobile imus reveal driver's identity from vehicle turns. *arXiv preprint arXiv:1710.04578*, 2017.

[104] D. Chen and K. G. Shin. Turnsmap: Enhancing driving safety at intersections with mobile crowdsensing and deep learning. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 3(3):78:1–78:22, Sept. 2019.

[105] H. Chin, H. Zabihi, S. Park, M. Y. Yi, and U. Lee. Watchout: Facilitating safe driving behaviors with social support. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '17, pages 2459–2465, New York, NY, USA, 2017. ACM.

[106] C. Corbett, J. Alexis, and L. Watkins. Who's driving you? In *Consumer Communications & Networking Conference (CCNC), 2018 15th IEEE Annual*, pages 1–4. IEEE, 2018.

[107] H. Dahlkamp, A. Kaehler, D. Stavens, S. Thrun, and G. Bradski. Self-supervised monocular road detection in desert terrain. In *Proceedings of Robotics: Science and Systems*, Philadelphia, USA, August 2006.

[108] J. Dai, J. Teng, X. Bai, Z. Shen, and D. Xuan. Mobile phone based drunk driving detection. In *International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, pages 1–8, March 2010.

[109] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR 2009*, pages 248–255. Ieee, 2009.

[110] P. Di Lena, S. Mirri, C. Prandi, P. Salomoni, and G. Delnevo. In-vehicle human machine interface: An approach to enhance eco-driving behaviors. In *Proceedings of the 2017 ACM Workshop on Interacting with Smart Objects*, SmartObject '17, pages 7–12. ACM, 2017.

[111] M. Diaz-Cabrera, P. Cerri, and P. Medici. Robust real-time traffic light detection and distance estimation using a single camera. *Expert Systems with Applications*, 42(8):3911 – 3923, 2015.

[112] C. Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, volume 4052, pages 1–12, Venice, Italy, July 2006. Springer Verlag.

[113] M. Enev, A. Takakuwa, K. Koscher, and T. Kohno. Automobile driver fingerprinting. In *Proceedings on Privacy Enhancing Technologies*, pages 34–50, 2016.

[114] M. Ester, H. peter Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[115] N. Fairfield and C. Urmson. Traffic light mapping and detection. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[116] B. Friedland. Treatment of bias in recursive filtering. *IEEE Transactions on Automatic Control*, 14(4):359–367, Aug 1969.

[117] K. Gade. The seven ways to find heading. *The Journal of Navigation*, 69(5):955–970, 2016.

[118] J. Galbally, S. Marcel, and J. Fierrez. Image quality assessment for fake biometric detection: Application to iris, fingerprint, and face recognition. *IEEE Transactions on Image Processing*, 23(2):710–724, 2014.

[119] R. K. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine*, 49(11):32–39, November 2011.

[120] T. Gebru, J. Krause, Y. Wang, D. Chen, J. Deng, E. L. Aiden, and L. Fei-Fei. Using deep learning and google street view to estimate the demographic makeup of neighborhoods across the united states. *Proceedings of the National Academy of Sciences*, 114(50):13108–13113, 2017.

[121] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[122] I. Google. Battery historian. `https://github.com/google/battery-historian`, 2017.

[123] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013.

[124] D. M. Gromet, H. Kunreuther, and R. P. Larrick. Political ideology affects energy-efficiency attitudes and choices. *Proceedings of the National Academy of Sciences*, 110(23):9314–9319, 2013.

[125] J. A. Gubner. *Probability and random processes for electrical and computer engineers*. Cambridge University Press, 2006.

[126] A. Halevy, P. Norvig, and F. Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.

[127] D. Hallac, A. Sharang, R. Stahlmann, A. Lamprecht, M. Huber, M. Roehder, R. Sosi, and J. Leskovec. Driver identification using automobile sensor data from a single turn. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 953–958, Nov 2016.

[128] K. A. Hallgren. Computing inter-rater reliability for observational data: an overview and tutorial. *Tutorials in quantitative methods for psychology*, 8(1):23, 2012.

[129] M. Hermans and B. Schrauwen. Training and analysing deep recurrent neural networks. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 190–198. Curran Associates, Inc., 2013.

[130] S. Hetrick. Examination of driver lane change behavior and the potential effectiveness of warning onset rules for lane change or “side” crash avoidance systems, 1997.

[131] M. Hirabayashi, A. Sujiwo, A. Monrroy, S. Kato, and M. Edahiro. Traffic light recognition using high-definition map features. *Robotics and Autonomous Systems*, 111:62–72, 2019.

[132] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[133] J.-H. Hong, B. Margines, and A. K. Dey. A smartphone-based sensing platform to model aggressive driving behaviors. In *Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems*, CHI ’14, pages 4047–4056, New York, NY, USA, 2014. ACM.

[134] S. Hu, L. Su, H. Liu, H. Wang, and T. F. Abdelzaher. Smartroad: Smartphone-based crowd sensing for traffic regulator detection and identification. *ACM Transactions on Sensor Network*, 11(4), July 2015.

[135] S. Jha. Characteristics and sources of noise and vibration and their control in motor cars. *Journal of Sound and Vibration*, 47(4):543 – 558, 1976.

[136] D. Johnson and M. Trivedi. Driving style recognition using a smartphone as a sensor platform. In *International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1609–1615, Oct 2011.

[137] C. J. Kahane and J. N. Dang. The long-term effect of abs in passenger cars and ltvs. Technical report, National Highway Traffic Safety Administration, 2009.

[138] G. Kar, S. Jain, M. Gruteser, J. Chen, F. Bai, and R. Govindan. Predriveid: pre-trip driver identification from in-vehicle data. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 2. ACM, 2017.

[139] Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[140] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014.

[141] B. I. Kwak, J. Woo, and H. K. Kim. Know your master: Driver profiling-based anti-theft method. In *Privacy, Security and Trust (PST)*, pages 1040–1045, June 2016.

[142] Z. C. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

[143] L. Liu, H. Li, J. Liu, C. Karatas, Y. Wang, M. Gruteser, Y. Chen, and R. P. Martin. Bigroad: Scaling road data acquisition for dependable self-driving. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '17, 2017.

[144] X. Liu, H. Mei, H. Lu, H. Kuang, and X. Ma. A vehicle steering recognition system based on low-cost smartphone sensors. *Sensors*, 17(3):633, 2017.

[145] M. V. Ly, S. Martin, and M. M. Trivedi. Driver classification and driving style recognition using inertial sensors. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 1040–1045, June 2013.

[146] F. Martinelli, F. Mercaldo, A. Orlando, V. Nardone, A. Santone, and A. K. Sangaiah. Human behavior characterization for driving style recognition in vehicle system. *Computers & Electrical Engineering*, 2018.

[147] E. Massaro, C. Ahn, C. Ratti, P. Santi, R. Stahlmann, A. Lamprecht, M. Roehder, and M. Huber. The car as an ambient sensing platform [point of view]. *Proceedings of the IEEE*, 105(1):3–7, 2017.

[148] T. Menard, J. Miller, M. Nowak, and D. Norris. Comparing the gps capabilities of the samsung galaxy s, motorola droid x, and the apple iphone for vehicle tracking using freesim mobile. In *IEEE Intelligent Transportation Systems (ITSC)*, pages 985–990, 2011.

[149] C. Miyajima, Y. Nishiwaki, K. Ozawa, T. Wakita, K. Itou, K. Takeda, and F. Itakura. Driver modeling based on driving behavior and its evaluation in driver identification. *Proceedings of the IEEE*, 95(2):427–437, Feb 2007.

[150] L. K. Nandam and T. D. Hess. Dynamic change of left turn phase sequence between time-of-day patterns-operational and safety impacts. *Institute of Transportation Engineers*, 2000.

[151] S. Narain, T. Vo-Huu, K. Block, and G. Noubir. Inferring User Routes and Locations Using Zero-permission Mobile Sensors. In *IEEE Symposium on Security and Privacy*, May. 2016.

[152] S. Nowak and S. Rüger. How reliable are annotations via crowdsourcing: a study about inter-annotator agreement for multi-label image annotation. In *Proceedings of the international conference on Multimedia information retrieval*, pages 557–566. ACM, 2010.

[153] M. Pal. Random forest classifier for remote sensing classification. *International Journal of Remote Sensing*, 26(1):217–222, 2005.

[154] V. M. Patel, R. Chellappa, D. Chandra, and B. Barbello. Continuous user authentication on mobile devices: Recent progress and remaining challenges. *IEEE Signal Processing Magazine*, 33(4):49–61, July 2016.

[155] M. D. Pesé, T. Stacer, C. A. Campos, E. Newberry, D. Chen, and K. G. Shin. Librecan: Automated can message translator. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, pages 2283–2300, New York, NY, USA, 2019. ACM.

[156] R. Ponziani. Turn signal usage rate results: A comprehensive field study of 12,000 observed turning vehicles. *SAE International Paper, 2012-01-0261*, 2012.

[157] H. Qiu, J. Chen, S. Jain, Y. Jiang, M. McCartney, G. Kar, F. Bai, D. Grimm, M. Gruteser, and R. Govindan. Towards robust vehicular context sensing. *IEEE Transactions on Vehicular Technology*, PP(99):1–1, 2017.

[158] L. Rabiner. On the use of autocorrelation analysis for pitch detection. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25(1):24–33, 1977.

[159] A. Riener and A. Ferscha. Supporting implicit human-to-vehicle interaction: Driver identification from sitting postures. In *The first annual international symposium on vehicular computing systems (isvcs 2008)*, page 10, 2008.

[160] E. Shi, Y. Niu, M. Jakobsson, and R. Chow. Implicit authentication through learning user behavior. In *Proceedings of the 13th International Conference on Information Security*, ISC'10, pages 99–113, Berlin, Heidelberg, 2011. Springer-Verlag.

[161] C. Sousedik and C. Busch. Presentation attack detection methods for fingerprint recognition systems: a survey. *IET Biometrics*, 3(4):219–233, 2014.

[162] F. Tahmasbi, Y. Wang, Y. Chen, and M. Gruteser. Poster: Your phone tells us the truth: Driver identification using smartphone on one turn. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, pages 762–764, 2018.

[163] T. Tanaka, K. Fujikake, Y. Yoshihara, T. Yonekawa, M. Inagami, H. Aoki, and H. Kanamori. Driving behavior improvement through driving support and review support from driver agent. In *Proceedings of the 6th International Conference on Human-Agent Interaction*, HAI '18, pages 36–44, New York, NY, USA, 2018. ACM.

[164] B. C. Tefft. American driving survey: 2015-2016. (research brief), 2018.

[165] T. Toledo and D. Zohar. Modeling duration of lane changes. *Transportation Research Record*, 1999(1):71–78, 2007.

[166] Traffic lane. `http://en.wikipedia.org/wiki/Lane`.

[167] G. Turin. An introduction to matched filters. *IRE transactions on Information Theory*, 6(3):311–329, 1960.

[168] Turning radius. `http://en.wikipedia.org/wiki/Turning_radius`.

[169] B. Wang, S. Panigrahi, M. Narsude, and A. Mohanty. Driver identification using vehicle telematics data. In *SAE Technical Paper*. AE International, 2017.

[170] Y. Wang, J. Yang, H. Liu, Y. Chen, M. Gruteser, and R. P. Martin. Sensing vehicle dynamics for determining driver phone use. In *Proc. of ACM MobiSys*. ACM, 2013.

[171] M. Weiser, R. Gold, and J. S. Brown. The origins of ubiquitous computing research at parc in the late 1980s. *IBM Systems Journal*, 38(4):693–696, 1999.

[172] J. S. Wijnands, J. Thompson, G. D. Aschwanden, and M. Stevenson. Identifying behavioural change among drivers using long short-term memory recurrent neural networks. *Transportation Research Part F: Traffic Psychology and Behaviour*, 53:34 – 49, 2018.

[173] S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[174] J. Yang, S. Sidhom, G. Chandrasekaran, T. Vu, H. Liu, N. Cecan, Y. Chen, M. Gruteser, and R. P. Martin. Sensing driver phone use with acoustic ranging through car speakers. *IEEE Transactions on Mobile Computing*, 11(9):1426–1440, Sept 2012.

[175] T. Yang. Networked control system: a brief survey. *IEE Proceedings - Control Theory and Applications*, 153:403–412(9), July 2006.

[176] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 351–360, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee.

[177] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *NIPS*, 2014.

[178] C.-W. You, N. D. Lane, F. Chen, R. Wang, Z. Chen, T. J. Bao, M. Montes-de Oca, Y. Cheng, M. Lin, L. Torresani, and A. T. Campbell. Carsafe app: Alerting drowsy and distracted drivers using dual cameras on smartphones. In *Proc. of ACM MobiSys*. ACM, 2013.

[179] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*, 2018.

[180] C. Zhang, M. Patel, S. Buthpitiya, K. Lyons, B. Harrison, and G. D. Abowd. Driver classification based on driving behaviors. In *Proceedings of the 21st International Conference on Intelligent User Interfaces*, IUI '16, pages 80–84, New York, NY, USA, 2016. ACM.

[181] P. Zhou, M. Li, and G. Shen. Use it free: Instantly knowing your phone attitude. In *Proc. of ACM Mobicom*. ACM, 2014.