CarSafe App: Alerting Drowsy and Distracted Drivers using Dual Cameras on Smartphones

Chuang-Wen You[†][*], Nicholas D. Lane[‡], Fanglin Chen[†], Rui Wang[†], Zhenyu Chen[†][⊔],Thomas J. Bao[†]
Martha Montes-de-Oca[⋈], Yuting Cheng[†], Mu Lin[†], Lorenzo Torresani[†], Andrew T. Campbell[†]
[†]Dartmouth College, [‡]Microsoft Research Asia, [*]Academia Sinica
[⊔]Chinese Academy of Sciences, [⋈]National Autonomous University of Mexico
cwyou@citi.sinica.edu.tw, niclane@microsoft.com, {chentc, ruiwang, zhenyu}@cs.dartmouth.edu
thomas.j.bao@dartmouth.edu, martham@unam.mx, {yutingc, linmu, lorenzo, campbell}@cs.dartmouth.edu

ABSTRACT

We present CarSafe, a new driver safety app for Android phones that detects and alerts drivers to dangerous driving conditions and behavior. It uses computer vision and machine learning algorithms on the phone to monitor and detect whether the driver is tired or distracted using the front-facing camera while at the same time tracking road conditions using the rear-facing camera. Today's smartphones do not, however, have the capability to process video streams from both the front and rear cameras simultaneously. In response, CarSafe uses a context-aware algorithm that switches between the two cameras while processing the data in real-time with the goal of minimizing missed events inside (e.g., drowsy driving) and outside of the car (e.g., tailgating). Camera switching means that CarSafe technically has a "blind spot" in the front or rear at any given time. To address this, CarSafe uses other embedded sensors on the phone (i.e., inertial sensors) to generate soft hints regarding potential blind spot dangers. We present the design and implementation of CarSafe and discuss its evaluation using results from a 12-driver field trial. Results from the CarSafe deployment are promising – CarSafe can infer a common set of dangerous driving behaviors and road conditions with an overall precision and recall of 83% and 75%, respectively. CarSafe is the first dual-camera sensing app for smartphones and represents a new disruptive technology because it provides similar advanced safety features otherwise only found in expensive top-end cars.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Design, Experimentation, Performance.

Keywords

Driver safety, dual-camera sensing, smartphone sensing.

1. INTRODUCTION

Driving while being tired or distracted is dangerous. In 2010, 3,092 people were killed and 416,000 injured in the United States alone during accidents directly attributed to distracted drivers [37]. Surprisingly, many people drive while being tired or drowsy [35] and according to experts, many drivers fail to recognize they are in a fatigued state. Tracking dangerous driving behavior can help raise drivers' awareness of their driving habits and associated risks, thus, helping reduce careless driving and promoting safe driving practices. Today's top-end cars come with a wealth of new safety features built-in. These include collision-avoidance, drowsy driver feedback (e.g., vibrating steering wheel), lane departure warning, lane weaving and pedestrian detection. By fitting advanced sensors into the vehicle (e.g., night cameras, radars, ultrasonic sensors), the car can infer dangerous driving behavior, such as drowsiness or distracted driving; some cars even trigger automatic steering when the car drifts into another lane or brake before getting dangerously close to the car in front. However, only a tiny percentage of cars on the road today have these driver alert systems; it will take a decade for this new technology to be commonplace in most cars across the globe. What do you do if you can't afford a top of the line car with advanced safety features?

We propose CarSafe, the first driver safety app that uses dual cameras on smartphones – shown conceptually in Figure 1 and in operation in Figure 2. CarSafe uses computer vision and machine learning algorithms on the phone to detect whether the driver is tired or distracted using the front-facing camera while at the same time tracking road conditions using the rear-facing camera. CarSafe focuses on five of the most commonly occurring dangerous driving events: drowsy driving, inattentive driving (e.g., when the driver is distracted and takes their eyes off the road), tailgating (i.e., getting too close to the car in front), lane weaving or drifting and ignoring blind spots during lane changes. CarSafe aims to mimic the safety features found in many of the top-end cars on the market today. However, the big advantage of CarSafe is that you do not need to drive a top-end car to get these safety features – all you need is a smartphone.

Several research projects are designing vision-based algorithms to detect drowsiness (using fixed mounted cameras in the car) or road conditions (using fixed or smartphone cameras [11]). These solutions usually detect driver states or road conditions limiting inference to either the road or driver side but not both. Today's smartphones do not, however, have the capability to process video streams from both the front and rear cameras simultaneously. To address this, CarSafe uses a context-based camera switching algorithm to schedule processing between two different camera classification pipelines at the "right time". The front camera pipeline

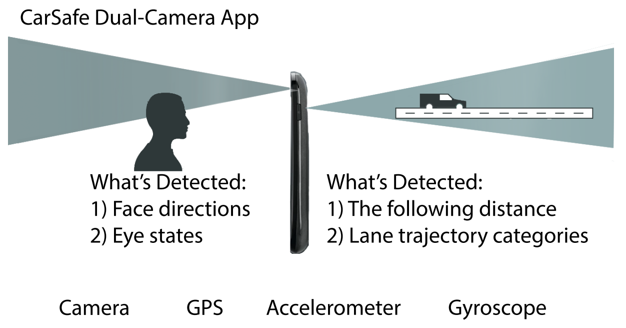


Figure 1: CarSafe exploits dual-camera sensing to track both driver and road-based events.

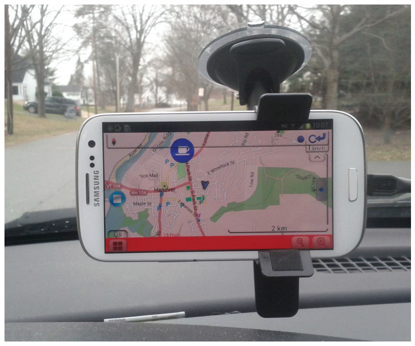


Figure 2: CarSafe, the first dual-camera sensing app, running on a Samsung Galaxy S3 mounted on the windshield of a car.

tracks the driver's head pose and direction as well as eyes and blinking rate as a means to infer drowsiness and distraction. If it detects drowsiness (or distraction), CarSafe alerts the driver by displaying a coffee cup icon (or an attention icon) on the phone's touch screen along with an audible alert. The rear camera pipeline monitors the distance between cars to determine if the driver is too close to the car in front as well as tracking lane change conditions. If it detects that the driver is too close to the car in front, a color status bar on the touch screen changes from green to red along with an audible alert. If CarSafe detects weaving across lanes on the road or other distracted or dangerous behavior, it alerts the driver by displaying an attention icon on the phone's touch screen along with an audible alert. The CarSafe UI shown in Figure 9 illustrates a number of these alerts.

CarSafe represents the first dual-camera sensing app, to the best of our knowledge, and therefore the design, insights and results provide a road map to others intent on building dual-camera apps in the future. The paper is structured as follows. §2 discusses the design considerations of implementing the CarSafe app on resource limited mobile phones that do not allow access to both cameras simultaneously. §3 presents the CarSafe architecture and detailed design based on three classification pipelines; that is, car, road, and driver classification pipelines. §4 describes the implementation details of CarSafe. This is followed in §5 by the evaluation of the CarSafe app. Results from a 12-person user study and system performance benchmarks are presented. The results look very promising – CarSafe can infer dangerous driving behavior and road conditions using context-driven camera switching with an overall precision and recall of 83% and 75%, respectively. Using blind spot hints and proximity information improves performance by up to 33% in terms of the F1 score. By adding intelligence into camera switching in comparison to a simple round robin approach boosts the F1 score by a further 16%. §6 discusses CarSafe in relation to related work. §7 presents some concluding remarks and future work.

2. DESIGN CONSIDERATIONS

In this section, we discuss the technical considerations that underpin the design of CarSafe while the detailed design is presented in §3.

2.1 Dangerous Driving Events

Drivers are faced with a multitude of road hazards and an increasing number of in-car distractions (e.g., music, conversation with passengers, phone calls, instrument panels, smartphone texting and browsing, etc.). CarSafe focuses on five of the most commonly occurring dangerous driving events [16].

Drowsy Driving (DD). Drivers who are fatigued are prone to episodes of microsleep [34]. Individuals are often unaware of these episodes; rather, people typically think they have been awake the whole time or have lost focus momentarily. As a result, drivers experiencing bouts of microsleep are at high risk of having an accident. While there has been work using static cameras [26] to infer microsleep, there has been no work on using smartphones to infer this with drivers. As a starting point, the smartphone's front camera should be able to monitor the prolonged and frequent blinks indicative of microsleep. There are many challenges to classifying microsleep including the diversity of people (e.g., glasses, facial expressions – such as smiling) and lighting conditions (e.g., overexposure, shadows).

Inattentive Driving (ID). Maintaining eye contact with the road is fundamental to safe driving. Again, by using the front camera of the phone, the head position of the driver can be used to determine an inattentive driving behavior when the driver is not looking at the road ahead. There is a broader set of distracted behavior that can also result in inattentive driving, such as, conversation with passengers, rubbernecking, texting, browsing, people watching and reading maps. Rather than attempting to infer specific scenarios, we treat all of these as examples of the driver being inattentive to the road; that is, not looking at the road ahead when the car is moving forward.

Tailgating (TG). Drivers should maintain a safe minimum distance with the car immediately ahead to ensure they can stop safely, if necessary. The danger level can be assessed by comparing the current following distance to an ideal safe distance based on DMV[1] guidelines (e.g.,[16]) and an estimate of the car speed. There are a number of commercial driving applications [11, 13] that use car-mounted smartphones and the rear camera to compute following distance and provide feedback to the driver (see §6); these applications are closed source and therefore it is not clear how they compute distances, in fact these applications often do not display the distance but a "following time" between cars. By developing this feature, our contribution is to offer a similar service but provide the technical details of how this is done – in addition we integrate this technique with a variety of others to provide more comprehensive protection.

Lane Weaving and Drifting (LW). Lane weaving happens when a driver performs one or more erratic lane changes. For example, frequent lane changing is an effort to maintain a high speed and

[1]Department of Motor Vehicles

avoid traffic congestion. Lane drifting is the inability of the driver to keep their vehicle within the lane markers. The CarSafe app should be capable of recognizing these unsafe car trajectories using both the inertial sensors and the rear camera of the phone when available. The rear camera monitors the car's trajectory by detecting and tracking lane markers. Similarly, inertial sensors observe changes in direction and acceleration. Top-end cars come with lane drifting and weaving detection as well as "real" blind spot monitoring (i.e., the car detects if another car is in a blind spot and flashes an icon). We aim to offer similar lane monitoring facilities using a commodity smartphone.

Careless Lane Change (CLC). Executing lane changes safely also requires a driver to check blind spots before proceeding. The driver does this by looking in the side and front mirrors of the car to check for unexpected vehicles. CarSafe should be capable of recognizing the head position of the driver using the phone's front camera, allowing the app to ensure the appropriate mirror checks are performed before each lane change. Lane changing itself can be detected using the rear camera and inertial sensors, as described above.

2.2 Dual Camera Processing

CarSafe fundamentally relies on the real-time processing of dual camera video streams. In what follows, we discuss the challenges and design considerations that arise.

Cross Stream Event Correlation. As shown in Figure 1 driving events can be independently monitored by each camera; for example, the front-facing camera detects the driver might be entering microsleep due to excessive long blinking; the rear-facing camera is watching the car ahead and is on the cusp of raising a tailgating alert. This is a likely event if a driver is entering microsleep. This example motivates the need for the simultaneously processing of video streams from the front and rear cameras. In addition, there may be strong correlation between events in and outside of the car, or causality: the driver is entering a drowsy state and because of that tailgating occurs. In contrast, there maybe no correlation between events on opposite sides of the camera streams – such as, the driver is attentive to the road but is driving too close to the car in front. In this example, there is no evidence of causality. Therefore, the app should be capable of fusing events detected from cameras and sensor readings to infer more complex behavior.

Limited Dual Camera Access. At the time of developing CarSafe, none of the mainstream smartphones provide simultaneous access to both the front and rear cameras. As a result, dual-camera sensing apps will be forced to switch periodically between the front and rear cameras. Motivated by this limitation, we conduct a series of experiments spanning a variety of high-end phone models and various manufacturers (viz. Nokia, Samsung, HTC, Apple). Not only did we verify that none of these platforms allow developers dual camera access but we also find that switching between cameras incurs a non-negligible delay. We refer to this delay as the *switching delay*. Table 1 presents the switching delay for the Samsung Galaxy S3 (Android 4.2), HTC One X (Android 4.2), Nokia Lumia 900 (Windows 7.5), iPhone 4S (iOS 5), and iPhone 5 (iOS 5) platforms. On average, delays across these platforms last 886 ms, with some platforms requiring more than 2000 ms to switch. Furthermore, we find the switching delay is asymmetric, and depends on the direction from which the switch is occurring (i.e., from the rear camera to the front camera or vice versa). All dual camera sensing apps must cope with the switching delay if they are to function on off-the-shelf hardware. In this paper, we implement CarSafe using the Android platform to investigate the challenges faced in building a dual camera app under a mainstream platform without assistance from specialized hardware, device drivers, or performing any OS hacking.

To further understand the reasons for this limitation, we study the source code of the Android Camera Architecture [3]. This framework employs three architectural layers to abstract away the complexity of directly manipulating camera hardware (e.g., via the Video4Linux (V4L) library [14]): (1) the application framework, (2) camera service, and (3) the hardware abstraction layer (HAL). By adopting a camera service layer to mediate between the application framework and the HAL, developers are isolated from the hardware. As a result, developers only need to interact with a standard Java interface. In contrast, platform engineers implementing the HAL on each smartphone require specialized hardware knowledge. This causes manufacturers to be resistant to significant architectural changes in existing HAL implementations. Because earlier smartphones only included a single camera, HALs are naturally designed with only single-camera access in mind. For example, the HAL for the Texas Instruments OMAP4460 chipset [17] used by the Samsung Galaxy Nexus limits the number of simultaneously connected camera streams to be one (i.e., the `MAX_SIMUL_CAMERAS_SUPPORTED` constant in the `CameraProperties.h` file is set to 1). Even though phone manufacturers have included multiple cameras along with the necessary hardware capabilities to process simultaneous streams, hardware vendors have yet to revise HAL implementations to expose this capability to developers. As a result, we find that if an app already has an opened camera stream, then all additional requests from an app to open additional camera streams (for example, through the Android Camera Service) are blocked at the HAL with runtime exceptions returned.

Phone manufacturers are beginning to react to this artificial dual-camera limitation [19]. For example, in April 2013 – after the CarSafe project was complete – Samsung announced support for dual camera access in the upcoming Samsung Galaxy S4. The Galaxy S4 will be equipped with Samsung's own Exynos 8-core processor, providing the necessary computational power to process multiple image streams at the same time. Samsung will redesign the HAL included with the Galaxy S4 to support dual camera access [19]. Because we design CarSafe to operate without native dual camera support it will operate even better on smartphones that offer zero switching delay (e.g., Galaxy S4). Furthermore, because the application framework layer remains unchanged, even under the S4, CarSafe will not require modifications to leverage new dual camera capabilities. CarSafe is an example of a dual camera sensing app that incorporates a unified implementation able to function correctly on both existing phones (e.g., Samsung Galaxy Nexus and S3) and future (e.g., Samsung Galaxy S4) dual-camera phones.

Blind Spot Processing. The fact that only one camera can be accessed at a time and the cost of switching – that is, the switching delay – results in blind spots and potentially missed events. Essentially, a switching solution implies that a dual camera app technically will have a blind spot in the front or rear at any given time.

Table 1: Camera switching overhead, including switching directions: front-to-rear (F-R) and rear-to-front (R-F).

Overhead / Model	F-R (ms)	R-F (ms)	Face detection (ms)
Nokia Lumia 900	804	2856.3	2032.5
Samsung Galaxy S3	519	774	301.2
HTC One X	1030	939	680.3
iPhone 4S	446	503	70.92
iPhone 5	467	529	58.48

How do we determine the best time to switch cameras (e.g., the system is satisfied that an event is unlikely on one camera and switches to the other)? Furthermore, if the system has to switch, can it "fill in" in some capacity while in a blind spot? One way forward is to explore the use of other embedded sensors that are "always-on", and not subject to the limitations of camera switching. For example, CarSafe should be able to exploit inertial sensors, such as accelerometers, gyroscopes and digital compasses, to figure out what is occurring in the blind spot at any given time. In essence, CarSafe should be able to use other embedded sensors to compensate when the camera is looking at the wrong place. In addition, these "blind spot hints" can be used to enable intelligent camera switching.

2.3 Real-Time Performance

Table 1 shows the per-frame processing time for the same set of phone models described earlier. When running a representative workload (i.e., performing face detection on the phone) for an image analyzing task on Android [10], iOS [12] and Windows Phone [9] platforms, the processing time per frame ranges between 58 ms and 2032 ms. In comparison, the processing time is just 33 ms per frame when simply capturing frames without running any workload. As the phone executes more complex operations, it will require longer processing time for each frame, and therefore can not achieve real-time processing. However, with the availability of a multicore processor on these high-end phones, the amount of delay should be able to be further reduced if we can run tasks across multiple cores simultaneously. We will discuss how to boost the performance of image processing pipelines later in §3.5.

3. CARSAFE DETAILED DESIGN

In this section, we present a detailed discussion of the CarSafe architecture and algorithms, as illustrated in Figure 3. We begin with an overview of the architecture.

3.1 Overview

The key components include (1) driver, car and road classification pipelines; (2) dangerous driving event engine; (3) context-driven camera switching; (4) multi-core computation planner; and, (5) user interface.

Classification Pipelines. CarSafe comprises three sets of classifiers to infer driver behavior, car events and road conditions. Each component is designed and implemented as a classification pipeline operating on a different set of phone sensors.

Driver Classification Pipeline. Frames from the front camera are used to recognize (1) face direction – the turning angle, needed to track driver attention; and, (2) eye state – open/closed events, required for drowsiness detection.

Road Classification Pipeline. Frames from the rear camera are used to estimate (1) the following distance; and, (2) lane trajectory categories (i.e., lane change, and weaving).

Car Classification Pipeline. The GPS and inertial sensors are used to (1) estimate speed, (2) detect turns and (3) recognize lane trajectory categories. CarSafe exploits the continuous availability of these always-on sensors to make inferences when "the camera is looking the wrong way"; for example, inertial sensors can infer lane weaving when the camera is "watching" the driver.

Dangerous Driving Event Engine. Recognizing dangerous driving events requires the synthesis of the individual sensor-based inferences from the driver, car and road classification pipelines as overviewed above. CarSafe performs this cross stream event correlation (as discussed in the §2.2) by encoding each category of dangerous driving behavior listed in §2.1 as one or more rules. Rules are developed based on best practices for safe and defensive driv-

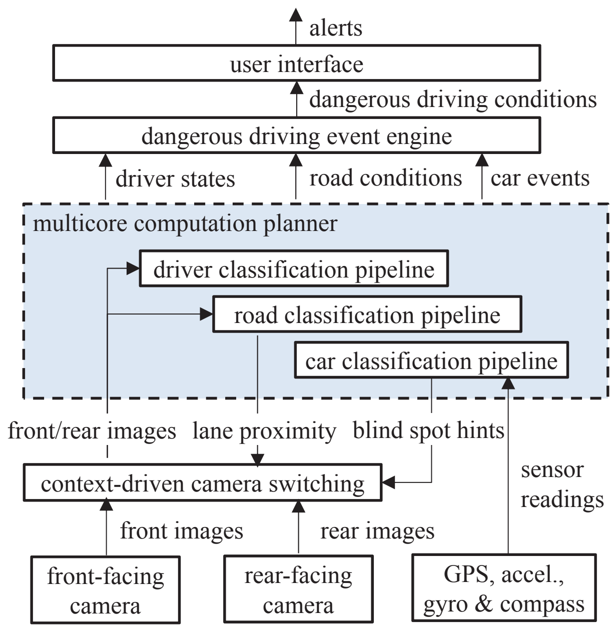

alerts
user interface
dangerous driving conditions
dangerous driving event engine
driver states
road conditions
car events
multicore computation planner
driver classification pipeline
road classification pipeline
car classification pipeline
front/rear images
lane proximity
blind spot hints
sensor readings
context-driven camera switching
front images
rear images
front-facing camera
rear-facing camera
GPS, accel., gyro & compass

Figure 3: The CarSafe architecture.

ing as outlined by DMVs [16, 8], traffic safety studies [7, 34] and through consultations with domain experts. The dangerous driving event engine evaluates each rule against the incoming stream of inferences (i.e., driver state, road conditions, and car events) made when sensor data is sampled.

Context-driven Camera Switching. Both driver and road conditions are subject to rapid and frequent changes demanding both the front and rear cameras to be continuously monitored. Because smartphones do not allow simultaneous access to both cameras CarSafe incorporates an adaptive context-driven camera switching policy. This mechanism is designed to minimize the number of events missed. It is also responsive to blind spot hints from the car classification pipeline.

Multi-core Computation Planner. Without carefully constructing and scheduling the computational workload necessary for CarSafe, user feedback can not be reliably generated in near real-time. As a result, for example – an alarm to alert a driver that they are frequently entering periods of microsleep may not be delivered in time. The planner enables CarSafe to more fully utilize the multi-core computational resources of the smartphone.

User Interface. Our CarSafe prototype can either be used by a driver as a stand-alone smartphone app, or as a transparent extension to an existing in-car app – for instance, car navigation. In all of these scenarios, the basic operation of CarSafe remains the same; whenever a dangerous driving event is detected a visual icon (e.g., a coffee cup indicating drowsy driving) is shown along with an audible alert. Icons are overlaid on the host app interface, as shown in Figure 9.

3.2 Classification Pipelines

We begin by describing the design of the three sensor-based classifiers used by CarSafe to track the state of the driver, road and car, respectively.

3.2.1 Driver Classification Pipeline

The driver classification pipeline, shown in Figure 4, comprises

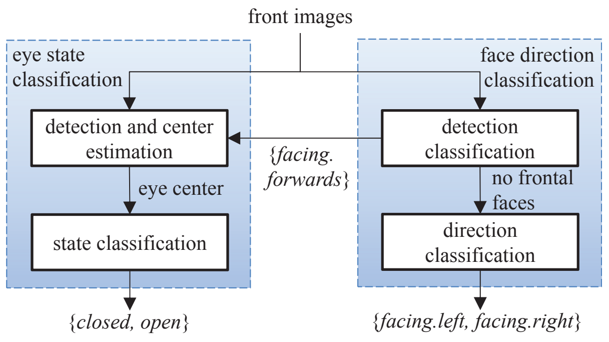


Figure 4: Data flow of driver classification pipeline.

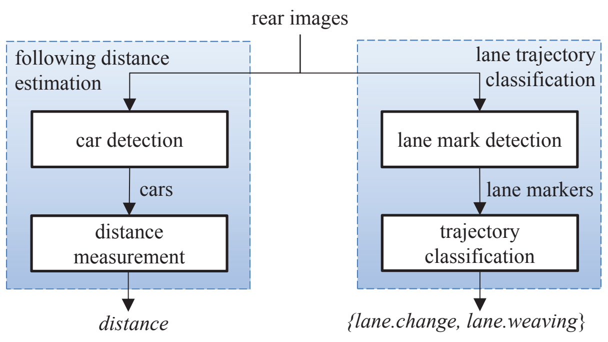

rear images
following distance estimation
car detection
cars
distance measurement
distance
lane trajectory classification
lane mark detection
lane markers
trajectory classification
{lane.change, lane.weaving}

Figure 5: Data flow of the road classification pipeline.

two components: (1) face direction classification, and (2) eye state classification.

Face Direction Classification. Inferring the direction of the driver's face is divided into two steps: (1) detection classification and (2) direction classification.

Detection Classification. Images from the front camera are scanned to find the relative position of the driver's face. The scanning procedure is an iterative process. At each iteration a selected region of the overall image is provided to a classifier that determines if a face is present, and if so the face direction is classified. Initially, the size of the region is set to 200 x 200 pixels. If no face is found, the region is made smaller with the process terminated once a minimum region size is reached (70 x 70 pixels in the CarSafe prototype).

Direction Classification. To classify the direction of the driver's face we use two steps: (1) Haar-like features [41] that are extracted from image regions provided by the detection stage, after which (2) inference is performed using an Adaboost decision-stump classifier [33]. Haar-like features highlight differences between object classes (in this case faces) by aggregating pixel intensities for rectangular regions. We adopt this classifier design (i.e., combination of feature and model type) because it has been demonstrated to be effective in recognizing a variety of object types, including faces. The classifier is a cascade of decision-stumps (i.e., two-level decision trees). We train this model to recognize four face related categories that include: (1) no face is present; or the driver's face is either (2) facing forwards (defined as *facing.forwards* events), towards the road; (3) facing to the left, defined as *facing.left* events (i.e., a $\geq 15^\circ$ rotation relative to facing directly forward); and, (4) facing to the right, defined as *facing.right* events (another $\geq 15^\circ$ rotation but this time to the right).

Eye State Classification. The classification of driver's eye states (i.e., open and closed states) requires two steps: (1) detection and center estimation and (2) state classification. This stage is only performed if the driver's head is recognized as facing the road during face direction classification.

Detection and Center Estimation. This process is initialized by receiving the pixel co-ordinates of a region within the image that includes only the face of the driver from the face direction classifier. Similarly, two even smaller regions that enclose the eyes are searched for by exploiting a commonly used active shape model (available from [1]). This model uses 76 MUCT[2] pre-defined visual landmarks [21] that are fit to the image to estimate an eye bounding region. To find the eye center we calculate the vector field of image gradients with respect to the pixel intensities. By summing up the inner products of image gradients over each point within the iris, the eye center can be detected by finding the largest value.

State Classification. Eye classification is a two stage process comprising: (1) the extraction of SURF[3] features [25], followed by (2) inference using an SVM[4] classifier [28]. We compute 64-component SURF features to represent the two states (viz. open and closed) of the eye. Extracting features requires knowledge of the eye position and the pixel co-ordinates of the eye center, both of which are determined in the prior stage of the pipeline. SURF features are provided to a binary two-class SVM that is trained to classify eyes as either being open or closed (defined as an *open* or *closed* event). We train the SVM using the BioID face database [5]. Due to a limited number of examples of closed eyes in this database we recruit 30 volunteers to provide supplementary training data. In total our training set contains 160 open eye images and 160 closed eye images.

3.2.2 *Road Classification Pipeline*

As illustrated in Figure 5, the road classification pipeline comprises two components: (1) following distance estimation and (2) lane trajectory classification.

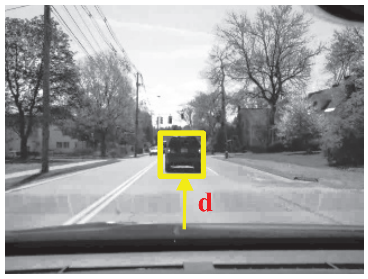
Following Distance Estimation. CarSafe estimates the following distance using a two stage process that begins with (1) the detection of the car immediately ahead, after which (2) the distance between the cars is estimated.

Car Detection. The recognition of a car within an image is essentially an identical process to the one performed by the face direction classifier; the iterative scanning process performed to detect cars is the same, as is the design of our car classifier (i.e., choice of features and classification model). In our prior work, WalkSafe [43], we verified this particular detector design is effective in recognizing cars. To train this classifier we use a labeled dataset where: positive examples (i.e., images containing cars captured from the rear) are sourced from the MIT CBCL[5] car dataset [38], and negative examples (i.e., images of cars taken at different angles along with various images containing road and urban settings) are collected from Google StreetView [20].

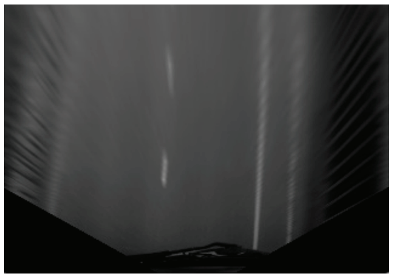
Distance Estimation. By applying a pin-hole camera projection we can estimate the following distance based on the pixel distance captured by the rear-camera image, such as Figure 6(a). To perform this projection requires the pixel co-ordinates within the image of

[2]Milborrow / University of Cape Town

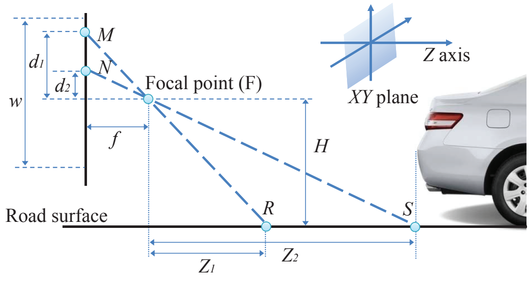
[3]Speeded Up Robust Features
[4]Support Vector Machine
[5]Center for Biological & Computational Learning



(a) Distance measurement



(b) Bird's-eye view image



(c) Pinhole model

Figure 6: Following distance measurement and lane detection steps. (a) The detected following distance (*d*, as indicated by the yellow arrow) in the gray-scale image. (b) The Bird's-eye view image is generated by applying inverse perspective mapping on the original gray-scaled image. (c) A pinhole model transforms in-image pixel distances between cars to real following distances.

(1) the car ahead and (2) the driver's car. The first co-ordinate pair (the car ahead) is provided by the previously described car detection stage; the second pair is detected with a simple threshold-based edge detector that identifies the front edge of the driver's car visible at the bottom of each image.

Figure 6(c) shows the pin-hole camera projection we perform. This figure illustrates how any road distance Z (e.g., Z_1 and Z_2) in the camera view is projected to an image pixel distance d (e.g., d_1 and d_2). As a result, the following distance can be computed by,

$$Z_2 - Z_1 = (\frac{1}{d_2} - \frac{1}{d_1}) \times f \times H \tag{1}$$

where Z_1 and Z_2 are the real-world distances between the smartphone and the front of the driver's car and the car immediately ahead respectively; d_1 and d_2 are the in-image pixel distance equivalents of Z_1 and Z_2; f is the rear-camera focal length; and finally, H is the smartphone mounting height.

Lane Trajectory Classification. This is a two stage process in which (1) lane markings are detected and tracked across multiple frames, after which (2) a classifier determines the trajectory category.

Lane Marker Detection. Detection of lane markers is performed in three stages: (1) perspective transformation, (2) vertical-edge detection, and (3) line detection. First, the perspective of each rear-camera image is transformed into a "bird's-eye view" by applying inverse perspective mapping [24]. The effect of the transformation can be seen by comparing Figure 6(a) and Figure 6(b). Second, possible lane marker edges are identified by a pixel-based search of the region of road immediately in front of the driver's car. For each pixel we compare it's gray-scale pixel value ($g(x, y)$) with that of pixels ($g(x - m, y)$ and $g(x + m, y)$) horizontally adjacent, where x and y are pixel co-ordinates and m is the lane marker width. As a result of this pixel search, we generate an edge-map in which pixels of potential lane markers are set as white and the remaining pixels as black. Third, we apply a Hough-line transformation [32] to the edge-map that will identify line segments and select lane markers. The Hough-line transformation we use first applies a Canny edge detector [29] before voting on line candidates using the RANdom SAmple Consensus (RANSAC) algorithm [32].

Trajectory Classification. Lane trajectory categories (including lane change (defined as *lane.change* events), and lane weaving (defined as *lane.weaving* events)) are distinguished by statistics that track lane marker crossing events. Crossing events are defined as when the intersection of the detected lane marker and the lower edge of the smartphone screen pass by the vertical center line of a frame. A decision tree [27] is trained based on (1) the duration and (2) the frequency of crossing events. We train the classifier using images collected during controlled driving maneuvers (see §5.1).

3.2.3 *Car Classification Pipeline*

Figure 7 shows the three components of the car classification pipeline: (1) speed estimation, (2) turn detection, and (3) lane trajectory classification.

Speed Estimation. Estimates of the vehicle's speed are based on time-series GPS positions. We first project the geodetic co-ordinates from the GPS (i.e., latitude and longitude) into a cartesian co-ordinate system. Speed is then estimated by dividing the sum of the distances between pairs of successive co-ordinates spanning a sliding window of five seconds.

Turn Detection. Similar to speed estimation, car turns are detected by observing significant changes in direction from time-series GPS positions. Again, GPS positions are first converted into cartesian co-ordinates after which vectors are formed from successive co-ordinates. Turns are detected by examining the change in angle between successive vectors within a sliding window of 10 seconds. A vehicle turn is inferred if the average change in angle exceeds a threshold, set to $\approx 50^\circ$ in our experiments. The relative direction of the turn (i.e., left- or right-hand turn) is based on the polar coordinates of the average vector change. If the change in angle is positive the turn is to the right (defined as *turn.right* events) otherwise the turn is to the left (defined as *turn.left* events).

Trajectory Classification. This classifier recognizes the same set of classes as the road classification pipeline; however, it uses a completely different set of sensors – inertial sensor readings rather than the rear camera.

From three inertial sensors – the accelerometer, gyroscope and compass – a series of features are extracted that capture distinctive characteristics of the car trajectory as it moves within and between

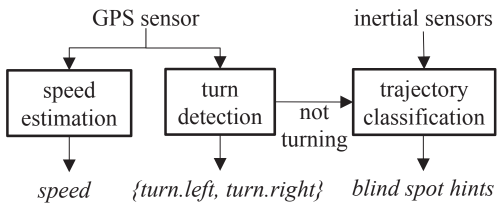


Figure 7: Data flow of the car classification pipeline.

the road lanes. We adopt a feature set and extraction algorithms previously developed in our own prior work [36]. Specifically, we use three time-domain features (mean, variance and mean-crossing rate) and five frequency-domain features (peak frequency, spectral entropy, spectrum sub-band energy, sub-band energy ratio, and spectrum correlation). These eight features are extracted for each sensor resulting in a 75-element feature vector. Prior experiments [36] have proven such features are effective in classifying physical activities (e.g., walking, running, standing) that have distinct time-series patterns within inertial sensor data – we now apply this same feature set to classifying car trajectories. In addition, the extraction algorithms used from [36] are robust to the precise position and orientation of the phone within the car.

Classification is performed using a binary bayesian classifier that represents classes as a multi-variate gaussian [27]. Two classes of trajectories are recognized – one "lane change/weaving" class for both types of lane trajectories (viz. lane change and weaving) and one "other" class that represents all other trajectories where the car largely remains in the same lane. Each feature vector is computed using a sliding window of 1.5 seconds of raw data (≈ 128 raw data segments). Classification is performed over each input feature vector to recognize the current lane trajectory as being either the "lane change/weaving" or "other" class. To train the classifier, we use the driving data (see §5.1 for details) and manually label each data segment. Inferences (e.g., blind spot hints) can be use as input to context-driven camera switching.

3.3 Dangerous Driving Event Engine

CarSafe recognizes five types of dangerous driving events (detailed in §2.1). For each type, we describe the series of rules used to interpret the low-level state inferred by the classifiers described in §3.2 that allow dangerous events to be detected.

Drowsy Driving (DD). We adopt a standard metric for measuring alertness, PERcentage of CLOSure of the eyelid – called PERCLOS [44], which more formally represents the proportion of time within one minute that eyes are at least 80% closed [34]. This driver state information is provided by the driver classification pipeline detailed in §3.2.1. CarSafe continuously computes PERCLOS and declares the driver "drowsy" if PERCLOS exceeds a threshold (28%) guided by prior studies [44].

Inattentive Driving (ID). Two types of inattentive driving are monitored by CarSafe. In the first case, the output of the face direction classifier is tracked. If the driver's face is not facing forward for longer than three seconds while the car is moving forward (i.e., while a positive speed is reported by the car classification pipeline) and not turning as reported by the turn detector (also reported by car classification pipeline) then a dangerous driving event is inferred. We use a three second threshold based on the guidelines provided by [7]. In the second case, we monitor the turn detector. Each time a turn is detected the historical output of the face direction classifier is checked. If there is no a head turn corresponding to a car turning event then the driver did not check that the road is clear before turning – as a result, a dangerous event is inferred.

Tailgating (TG). We adopt the calculation for a minimum following distance from [16], which relies on the speed estimation made by the car classification pipeline. This estimate of a "safe following distance" is compared to the estimated current following distance also provided by the car classification pipeline. If CarSafe determines the safe following distance is not respected for a period longer than three seconds a dangerous driving event is inferred.

Lane Weaving and Drifting (LW). Detecting lane weaving or drifting relies on the trajectory classifier contained within the road classification pipeline. CarSafe reports a dangerous driving event if the classifier infers either lane weaving or lane drifting continuously for longer than two seconds, which would be significantly longer than the typical duration of a lane change maneuver [39].

Careless Lane Change (CLC). Each time the trajectory classifier associated with the road classification pipeline determines a lane change event has occurred the recent inferences made by face direction classification are examined. If there is no head turn corresponding to a lane change event (occurring within ϵ seconds prior to the lane change event detection) then a dangerous driving event is inferred. We set ϵ to three seconds based on the road safety findings provided in [31].

Algorithm 1: Context-driven Camera Switching

```
Input: $B_{min}$, $F_{min}$, $B_{add}$, and $F_{add}$
Output: Estimated time $T_f$ or $T_b$
Connecting to the front camera initially ($CurCamera = Front$);
while TRUE do
    Waiting for the next frame from the $CurCamera$;
    Processing the received frame;
    if Close to a lane marker && $CurCamera == Rear$ then
        $switchTime = currentTime + 2\ sec$;
    if ((Received a blind spot hint) && $CurCamera == Front$) || (switchTime ≥ currentTime) then
        if $CurCamera == Front$ then
            $CurCamera = Rear$;
            Estimate $T_b$ based on Eq. 2;
            $switchTime = currentTime + T_s^f + T_b$;
        else
            $CurCamera = Front$;
            Estimate $T_f$ based on Eq. 3;
            $switchTime = currentTime + T_s^b + T_f$;
        Switch to the other camera;
```

3.4 Context-driven Camera Switching

In what follows, we present the design of our context-driven camera switching.

Switching Algorithm. We provide the intuition that underpins the design of our camera switching mechanism; a formal description is provided in Algorithm 1.

Primarily, the switching between the front and rear cameras is regulated by two time intervals (i.e., T_e^f, and T_e^b). Each interval is associated with one of the two cameras. Once the time interval for a particular camera expires CarSafe switches to the other camera. The pair of time intervals are set by an event arrival prediction process, which seeks to dynamically adjust the time allocated to each camera based on the current driver, road or car conditions. For example, if the system determines that the driver is likely to enter a dangerously drowsy state more time is allocated to the front camera to monitor this situation closely. Figure 8 presents an example of the common case operation of the camera switching mechanism.

Scheduled camera switching based on a pair of timers is supplemented with temporary pre-emption of the switching plan driven by discrete events – we refer to this behavior as the pre-emption of the switching plan. For example, if the driver starts to drive close to the lane markers then a planned switch to the front camera can be delayed to allow further monitoring of the lane marker situation.

Event Arrival Prediction. We use two separate predictors that target two event types: (1) front-camera events, which focus on driver states; and (2) rear-camera events, which focus on car events and road conditions.

Front-Camera Events. The driver state event with the shortest time-scale is driver drowsiness (i.e., blink duration). We find for this reason the prediction of other driver state events are not needed and instead we focus on predicting driver drowsiness only. We again use PERCLOS to quantify drowsiness; the likelihood that a driver is drowsy increases inline with increases in PERCLOS. As a result, the time allocated to the rear camera is proportional to the most recent PERCLOS. More formally,

$$\mathrm{T}_e^f = \begin{cases} B_{add} \cdot \left(\frac{D_{TH} - \mathtt{PERCLOS}}{D_{TH}}\right) + B_{min}, & \text{if } \mathtt{PERCLOS} < D_{TH} \\ B_{min}, & \text{otherwise} \end{cases} \tag{2}$$

where D_{TH} is the threshold for PERCLOS, that determines if the driver is in a drowsy state; B_{add} is a scaler that regulates the relationship between PERCLOS and T_e^f; finally, B_{min} sets a floor for the prediction of T_e^f – to cope with the switching delay between cameras, B_{min} must be be greater than the switching delay ($\mathrm{T}_s^f + \mathrm{T}_s^b$).

Rear-Camera Events. Similar to the design of front camera event prediction, rear camera events also focus on a single key event type; that is, the following distance between cars. When the following distance is decreasing, we predict when this distance will be equal to – or smaller than – the minimum safe following distance (detailed in §3.3). In cases where the following distance is constant or increasing we enforce a maximum time between expected events. More formally,

$$\mathrm{T}_e^r = \begin{cases} F_{add} + F_{min}, & \text{if } \frac{D - D_s}{V} > (F_{add} + F_{min}) \\ max(\frac{D - D_s}{V}, F_{min}\}, & \text{otherwise} \end{cases} \tag{3}$$

where D_s is the minimum safe following distance; D is the current following distance, as estimated by CarSafe; F_{add} controls the varying range of T_e^r – to limit the maximum value of T_e^r when CarSafe detects no front cars and F_{min} sets a floor for the prediction of T_e^r. Similarly, F_{min} should be set greater than the switching delay ($\mathrm{T}_s^f + \mathrm{T}_s^b$).

Switching Plan Pre-emption. Planned switching events can be pre-empted by hints from sensors or current car context. We now describe two examples of pre-emption due to (1) proximity to lane markers and (2) expected trajectory changes based on sensor data observations.

Lane Proximity. When the car is close to lane markers a lane event (e.g., lane change) is more likely to occur. As a result, we introduce the following camera switching pre-emption heuristic. If the car is closer than 0.6 meters from the nearest lane marker then any planned camera switching is delayed (≈ 2.0 seconds), allowing an immediately occurring lane event to be captured.

Lane Trajectory – Blind Spot Hints. If lane trajectory changes happen while the front camera is active (rather than the rear camera), an event, such as a lane change or weaving, can go unobserved (i.e., missed events). As a result, we incorporate another pre-emption heuristic based on blind spot hints from the always-on inertial sensors, as discussed in §3.2.3. If the trajectory classifier (which uses blind spot hints) determines that changes in the car trajectory occur then the rear camera is immediately activated. By switching to the rear camera the road classification pipeline can immediately verify the car trajectory inference, after which camera switching returns to the timer based policy.

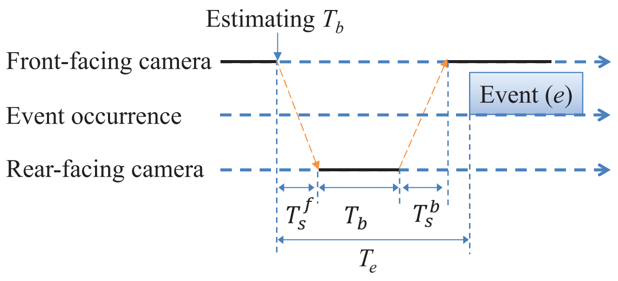


Figure 8: Context-driven Camera Switching algorithm in action. Here, an event (e) is monitored with the rear-facing camera. Although switching occurs roughly at the same time the event begins it is only monitored once the switching delay overhead (T_s^f) has past.

3.5 Multi-core Computation Planner

We determined experimentally in §2.3 that processing dual image streams from the front and rear cameras can overwhelm the smartphone's computational resources. To maintain near real-time user feedback CarSafe relies on a computational planner, which aims to effectively leverage the multi-core architecture of new smartphones to perform classification, as discussed in §3.2. The multi-core computation planner is comprised of three components: (1) dispatcher, (2) queue manager, and (3) de-multiplexer.

Dispatcher. A pool of threads is maintained by dispatcher. Each thread contains one single classifier, of the three described in §3.2. The dispatcher allocates new camera frames or other data to unused threads. Because frames do not have dependencies with other frames being processed, the allocation of data to threads is greatly simplified. For example, multiple threads each running the same road classification pipeline can operate on different captured image frames.

Queue Manager. The dispatcher may find no available threads when new data arrives to be processed. In such cases this data is placed in a queue to wait until a thread becomes available. When a thread does become available the most recent data (e.g., image frame) is used rather than the data that has been in the queue the longest. This allows the most recent data to always be processed as soon as possible. Any frames waiting that are older than the one selected are dropped from the queue and go unprocessed.

De-multiplexer. The processing time for frames is variable, even for identical classifiers. As a result, it is possible for the output of frame processing to arrive out of order. In other words, a new frame may be completed before an older frame even though the new frame may have started processing later than the older frame. To cope with this problem each frame is timestamped with the time when processing began. Upon computation being completed the results of multiple frames can be re-ordered based on these timestamps.

3.6 User Interface

Figure 9 illustrates CarSafe integrated into OsmAnd [18], an open-source car navigation application. The user interface overlays simple icons to the map screen that correspond to particular dangerous driving events. A different method is used for the following distance alert. A small rectangular border is placed around the conventional GUI. This frame changes color based on the following distance; that is, green for safe and red for unsafe.

CarSafe can also operate independently as a stand-alone application. The same GUI events and icons are displayed along with con-

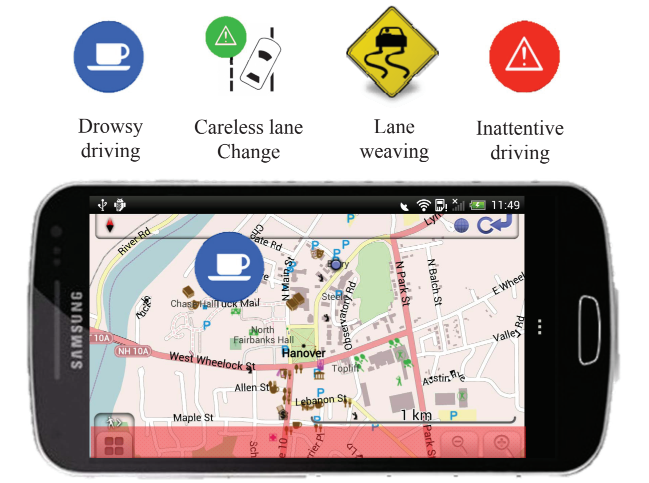


Figure 9: The example UI screenshot of CarSafe which indicates drowsy driving and tailgating conditions. The red rectangle at the bottom edge of the screen indicates the tailgating condition. The top row shows four icons representing for the rest of dangerous driving conditions.

tinually updating driver safety statistics (e.g., total time the driver has been inattentive of the road).

4. CARSAFE IMPLEMENTATION

We implement CarSafe on the multi-core Galaxy S3 Android phone. The driver, road and car classification pipelines, which represent the most computationally demanding modules, are written in C and C++ based on the OpenCV libraries and interfaced with Java using JNI wrappers. Other CafeSafe architectural components (viz. dangerous driving event engine, context-driven camera switching, and multi-core computation planner) are implemented using pure Java. To validate the prototype, we integrate the CarSafe modules with the OsmAnd navigation app. Because of privacy concerns, only Android programs running in the foreground can capture images from a camera through an Android image preview surface (e.g., SurfaceView). However, by overlaying an "invisible" dedicated drawing surface (shrinking it an extremely small size) on top of the original view hierarchy of OsmAnd, CarSafe modules can be easily embedded into, and seamlessly work with, OsmAnd without modifying the original navigation UI (e.g., the navigation map in Figure 9). The embedded drawing surface can receive preview images from either the front or rear camera at a rate of 30 fps. To boost the frame processing performance, we first set the resolution of preview images to 320 x 240 pixels. Because CarSafe classification pipelines represent computationally demanding models, single core phones have difficulty processing frames in real-time. To gain better performance we exploit new multi-core phones and target our first implementation to the Samsung Galaxy S3 with a 1.4 GHz quad-core Cortex-A9 CPU. We replicate the complete processing pipelines for the three classification pipelines (viz. driver, road and car classifiers) across four parallel threads on multiple CPU cores. When CarSafe is fully operational, the CPU usage is 64% for the Samsung Galaxy S3 (four CPU cores). While this represents a high continuous load on the phone, we argue that increased cores and advances in the development of more efficient computer vision and machine learning algorithms will allow the CarSafe pipelines to run at higher frame rates while riding the technology curve of more and more cores.

5. EVALUATION

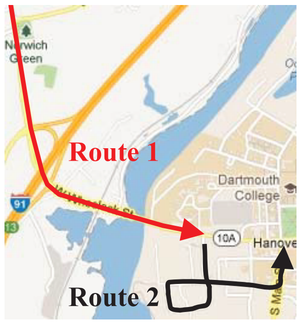
In this section, we evaluate CarSafe under real-world conditions where people use the application in the wild. Specifically, we discuss results from a user study where participants use CarSafe while commuting around the Hanover (New Hampshire, USA) area. In addition, we present results from a set of system performance benchmarks. Our findings include: (1) CarSafe can detect a range of dangerous driving conditions with acceptable levels of accuracy; (2) CarSafe's classification pipelines cope with a majority of everyday road conditions; (3) context-based camera switching is effective, and allows CarSafe to operate on existing phones given that today's phones do not allow simultaneous access to both cameras; and, (4) the multi-core computation planner improves system utilization when supporting the CarSafe workload, enabling higher frame rate throughput.

5.1 Datasets

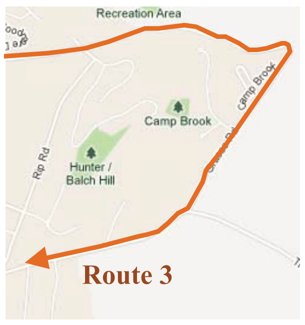
Collecting datasets to adequately evaluate CarSafe is challenging. This is because dangerous driving events are not guaranteed to happen during everyday driving for small scale trials of the app. In contrast, car manufacturers go to great lengths to ensure that there is a strong likelihood of observing dangerous driving events when evaluating new car safety technologies. For example, scientists working for Mercedes-Benz [15] evaluating safety features (e.g., weaving, drowsy driving) use specialized cars with dual steering wheels where participants of these drowsy driving studies are physically tired prior to the start of the experiment or simulation, hence increasing the probability of dangerous driving events occurring. Such an approach is beyond our means. As mentioned, only collecting data from normal/routine driving experiences is not viable on its own, simply because we can not accumulate enough examples of poor or dangerous driving to fully evaluate CarSafe. Furthermore, it would be irresponsible to run an experiment that promoted dangerous behavior without taking the sort of measures that the car manufacturers take.

For these reasons, we evaluate CarSafe using two distinct experiments and datasets: that is (1) *controlled car maneuvers*, where we safely "stage" dangerous driving events under controlled conditions using six study participants (6 males) – each driver is accompanied by a "co-pilot" who orchestrates the controlled maneuvers only when external conditions on the road and around the car are safe (e.g., where the co-pilot instructs the driver to weave when no other cars are on the road); and (2) *normal daily driving*, which only contains data collected during drivers' (using six study participants - 5 males and 1 female) everyday driving routines (e.g., commuting to and from work, going to the stores, etc.) without any controlled maneuvers or participation of a co-pilot. Note, that while participants in the controlled car maneuvers experiments all conduct the same set of controlled maneuvers they also conduct their normal driving between controlled maneuvers. We argue that this combination of experiments provide the necessary driving events required to evaluate CarSafe while minimizing the risks to participants.

We recruit a total of 12 participants (11 males and 1 female) with ages ranging from 23 to 53 years to collect their driving data under controlled car maneuvers and normal daily driving conditions. There are 6 participants in each of the study groups – 6 in the controlled car maneuvers group and 6 in the normal daily driving group. Participants are comprised of undergraduates, graduates, postdoctoral researchers, and faculty members of Computer Sci-



(a) Routes 1 and 2



(b) Route 3

Figure 10: Routes for the controlled car maneuvers dataset.

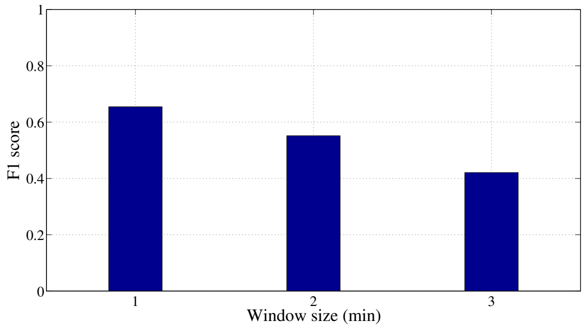


Figure 11: The F1 scores of the drowsy driving detection using different detection window sizes.

ence, Dartmouth College. A wide variety of different cars are used in the study including but not limited to: Dodge Stratus, Volkswagen Tiguan, Honda Civic, Subaru Forester, Subaru Outback, Nissan Camry, Volvo 850 and Mazda Familia. Both datasets combine to provide 300 minutes of driving data and include a total of 164 dangerous driving events. Under normal daily driving conditions, participants tend to drive their cars carefully, and therefore perform few dangerous driving events. Among these events, 121 dangerous driving events come from the controlled dataset and 43 events come from the normal daily driving dataset. The most common dangerous driving event found in the normal daily driving dataset is tailgating, where 22 tailgating events are detected.

Controlled Car Maneuvers Dataset. We recruit six participants to perform the role of drivers during our controlled experiments. All maneuvers are staged in one of three pre-defined routes (see Figure 10) selected for their lack of traffic. We use the participant's own vehicle during experiments, which is instrumented with three smartphones – one runs CarSafe, and two others capture frames from either the front or rear camera as well as capture sensor data (i.e., GPS, accelerometer, compass, gyroscope). After the experiments are completed we manually label dangerous driving events by replaying recorded video and segmenting the sensor and video streams as required. The labeling of such video is laborious, it requires multiple passes by different researchers to verify the ground-truth driving events are correct. Subjects repeat each of the following four maneuvers four times.

Scenario #1 (Careless lane change). Participants perform lane changes without checking their blind spots (i.e., not turning their heads to check the lane is clear). This scenario is performed along route 1 shown in Figure 10(a).

Scenario #2 (Tailgating). Participants drive along route 3 (shown in Figure 10(b)) behind another car driven by a researcher involved in the CarSafe project. The subject then periodically drives close to the car in front.

Scenario #3 (Driving while drowsy and lane weaving). Participants follow route 3, as shown in Figure 10(b). During the first half of the route, we ask drivers to periodically blink for an increasing period of time between 1 and 5 seconds, emulating microsleep. The other maneuver requires the drivers to weave between lanes erratically during the second part of the route.

Scenario #4 (Inattentive driving). While following route 2 (shown Figure 10(a)) participants are asked to periodically look away from the road while driving, as well as during left and right stationary turns.

Daily Driving Dataset. Six participants (5 males, 1 female) who drive daily are recruited to provide completely natural driving data that occurs as part of their everyday routine. Typically, most data comes from the subject's daily commute between home and work. We instrument the participants' cars just as we did for the prior dataset; that is, three smartphones are installed in each car, one running CarSafe and two others constantly capturing data; specifically, from the front- and rear-facing cameras. All videos from the front- and rear-facing cameras are manually labeled with ground-truth events which include all driving maneuvers, both normal and dangerous.

5.2 Overall CarSafe Accuracy

The key metric in CarSafe performance is its ability to detect instances of dangerous driving under real-world conditions. Table 2 provides the precision (*PR*) and recall (*RC*) results across all tested dangerous driving scenarios – including both controlled and normal daily driving groups. We find the mean precision and recall for all scenarios are 83% and 75%, respectively. Among these dangerous conditions, the cases related to detecting drowsy driving have a lower precision of 60%. To understand the cause of these particular false positive cases, we first analyze how the detection window size impacts the accuracy of the drowsy driving detection as follows.

Window Size Tuning – Drowsy Driving Detection. To determine the optimal value of the detection window size (described in §3.3), we develop a computer based simulator that includes the same CarSafe implementation that runs on Android smartphones (due to the portability of Java and by cross-compiling C++). We replay, through the simulator, the continuous image streams from the raw front- and rear-facing cameras captured during our experiments described earlier, along with all collected sensor data. As a result our simulator can test any possible switching behavior.

Table 2: The overall accuracy for detecting dangerous driving conditions. The columns provide: true positives (*TPs*), false positives (*FPs*), ground truth (*GT*), precision (*PR*), and recall (*RC*).

Condition	# of *TPs*	# of *FPs*	# of *GT*	*PR*	*RC*
Drowsy driving (DD)	18	12	25	0.60	0.72
Tailgating (TG)	62	8	78	0.89	0.79
Careless lane change (CLC)	12	2	14	0.86	0.86
Lane weaving (LW)	16	0	22	1.00	0.72
Inattentive driving (ID)	16	4	25	0.80	0.64
Overall	-	-	164	0.83	0.75

Figure 11 shows the F1 score[6] for drowsy driving detection when the window size is set to 1, 2, and 3 minutes. Before analyzing these window sizes, we run a pilot study to determine empirical thresholds of `PERCLOS` as 0.28, 0.31, 0.42 for detecting drowsy driving conditions using 1, 2, and 3 minute windows, respectively. Then, the F1 scores of different window sizes are obtained by running collected data using the simulator. We set the window size as 1 minute and the threshold for detecting drowsy driving conditions as 0.28, which results in a higher number of false positives. To further confirm the cause of these false positive cases, we manually analyze each false positive case and find that this is the result of poor lighting conditions (e.g., a shadowed or bleached out face) and facial expressions (e.g., smiling, squinting) that confuse the classifier, which are challenging scenarios for many vision-based algorithms. If we exclude these cases precision increases to 75%.

5.3 Driver, Road, and Car Classification Pipeline Benchmarks

We now evaluate each of the three classifiers responsible for monitoring driver, road and car states.

Driver Classification Pipeline. Table 3 reports the precision and recall for face direction events (i.e., *facing.right* (FR) and *facing.left* (FL)). We find for FR precision is 68% and recall is 68%; similarly, for FL the precision and recall are 79% and 88%, respectively. After studying these errors we identified two primary causes: (1) event loss (i.e., missing events) due to camera switching (39%); and, (2) "shallow face turnings", which are difficult to detect (17%). Errors in these face-based events (i.e., FR and FL) can also propagate to the process of recognizing driver inattention (i.e., ID) because these inferences are inputs used to detect this particular dangerous driving event. We find 71% of false negatives that occur when detecting driver inattention are due to errors in face direction inference. This is much larger than the fraction of false negatives caused by camera switching (14%) or due to failing to detect a car turn (14%). However, 66% of false positives (also for driver inattention) are due to undetected car turn events (discussed later) with the remainder caused by camera switching (33%).

As part of future work, we plan to incorporate other information (e.g., detecting the relative locations of the eyes in the detected face region) to filter out false positives when detecting face direction events.

To test the accuracy of eye state recognition within the driver classification pipeline we use a test dataset of front-camera frames. This dataset includes 1780 frames from six individuals. We hand label eye state for each frame. Table 4 provides the confusion matrix for the test dataset using the driver classifier. We find accuracy, precision and the false positive rates are 92%, 93% and 18%, respectively. The majority of false positives are due to challenging lighting conditions (5%) and reflections on driver's glasses (41%). These errors represent the limitations of vision algorithms when dealing with shadows, very bright light and reflections. However, the key use of eye states is for drowsy driver detection which uses an accumulation of prolonged blinks and eye closed events observed over a time window; therefore, the impact of errors in individual frames is diminished.

[6]F1 score is the harmonic mean of precision and recall [27]

Table 3: The overall accuracy for detecting individual low-level events. The rows are: *facing.right* (FR), *facing.left* (FL), *lane.change* (LC), *lane.weaving* (LW), *turn.right* (TR), and *turn.left* (TL).

Event	# of *TPs*	# of *FPs*	# of *GT*	*PR*	*RC*
Part 1: events detected from the driver classifier					
FR	21	10	31	0.68	0.68
FL	23	6	26	0.79	0.88
Part 2: events detected from the road classifier					
LC	21	1	24	0.95	0.88
LW	16	2	22	1.00	0.73
Part 3: events detected from the car classifier					
TR	31	0	35	1.00	0.89
TL	22	2	25	0.92	0.88
Overall	-	-	-	0.89	0.82

Table 4: Confusion matrix for the eye state classifying error.

# of frames		Detected	
		Open	*Closed*
Actual	*Open*	**1266**	41
	Closed	89	**384**

Road Classification Pipeline. Table 3 reports the detection accuracy of lane trajectory events (i.e., *lane.weaving* (LW), and *lane.change* (LC)) for the road classification pipeline. The table shows precision – 95% (LC) and 100% (LW) – is close to or higher than recall – 88% (LC) and 73% (LW). We investigate the cause of low recall for LW events and find this is due to the car being close to the lane markers for very brief periods during lane weaving – especially compared to the more controlled slower trajectory when performing a lane change (LC). To isolate the accuracy of lane marker detection state within the road classification pipeline (ignoring trajectory classification) we compile a test dataset of rear-camera images containing lane markers. This dataset contains 814 frames. For each frame we label (by hand) the position of the lane markers. We then provide these frames as input to the road classification pipeline directly. This experiment finds that lane markers are detected accurately in 85% of frames.

We perform a specific experiment to quantify the accuracy of following distance estimation. First, we instrument a stationary car with CarSafe. Next, another car is driven so it is positioned in front of the first car. We repeat the experiment 5 times with various distances between the two cars ranging between 5 and 35 meters. At each distance, manual measurements provide ground truth which is compared to the CarSafe distance estimate. Figure 12 reports the results from this experiment. We find that the mean error is 2.0 meters. We further find that following distance estimation is not possible beyond 35 meters. This is because the size of the front car in the image frame becomes too small and noisy for the estimation process to function correctly.

Car Classification Pipeline. Table 3 shows the turn detection performance of the car classification pipeline. Recognition of turn events (i.e., *turn.right* (TR) and *turn.left* (TL)) is 100% (precision) and 89% (recall) for TR events and 92% (precision) and 88% (recall) for TL events, respectively. We find all false positives (100%) are caused when driving along curved roads. The majority of false negative misclassifications (69%) are due to noisy GPS samples collected when the car stops for a long time before turning; other false negatives (31%) occur during turns that are $\leq 90^\circ$ (i.e., shallow turns). Our results show many of these errors may be removed by using existing map-matching algorithms [30] combined with the GPS. Table 5 provides a confusion matrix for the lane trajectory events inferred by the car classification pipeline. Mean precision and recall are 84% and 76%, respectively. Although both precision and recall appear to perform well, this component is primarily used to provide bind spot hints that are used to enable pre-emptive

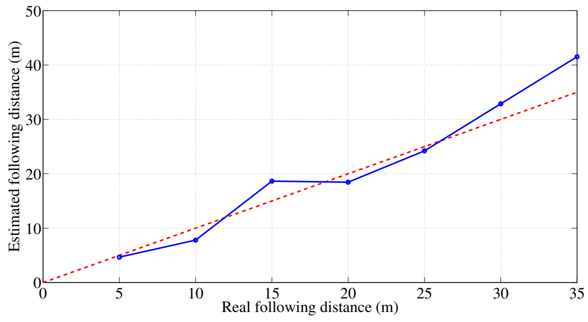


Figure 12: The accuracy of the estimated following distance (from 5 ~ 35 meters). The red dotted line indicates the ground truth of following distance, and the blue real line indicates the estimated following distance.

switching. For example, if the car classification pipeline detects a change in the car trajectory, CarSafe will immediately switch to the rear camera to verify the car trajectory inference.

5.4 Context-driven Camera Switching

In the following experiments, we perform parameter sensitivity analysis for context-driven camera switching and compare its performance to a baseline switching solution.

Methodology. Based on the same simulator described in subsection 5.2, we can test any possible switching behavior used during the experiment.

Baseline Switching Scheme. We compare our adaptive context-aware switching strategy (`carsafe`) to a static strategy (`baseline`) that gives equal time to both cameras in a simple round robing manner, but does not adjust to the environment (e.g., blind spot hints).

Parameter Tuning. Figure 13 shows the precision and recall for different parameter combinations for both `carsafe` and `baseline` switching schemes. During this experiment we perform a conventional grid parameter search with intervals of 1 sec within a parameter range of 1 to 30 seconds, for each parameter. The simulator provides both recall and precision values. Figure 13 shows that `baseline` performance is clustered at the bottom left hand side of the graph, compared to the top right hand side clustering for `carsafe`. In other words, across the majority of parameter combinations `carsafe` outperforms `baseline` – on average this margin is 25% (precision) and 14% (recall). By consulting Figure 13 we find for `carsafe` the optimal mean precision and recall (83% and 75%) respectively, is achieved when $\{F_{min}, B_{min}, F_{add}, B_{add}\}$ are set to 4, 4, 1, and 1 seconds. This combination represents the equilibrium when balancing two competing factors: (1) shorter switching times allow for more frequent updates from both

Table 5: Confusion matrix for detecting lane change/weaving through the car classifier. "Lane change/weaving" class represents either lane change or weaving trajectory while the "other" class represents all other trajectories where the car largely remains in the same lane.

# of data segments		Detected	
		Lane change / weaving	*Other*
Actual	*Lane change / weaving*	**190**	30
	Other	109	**1127**

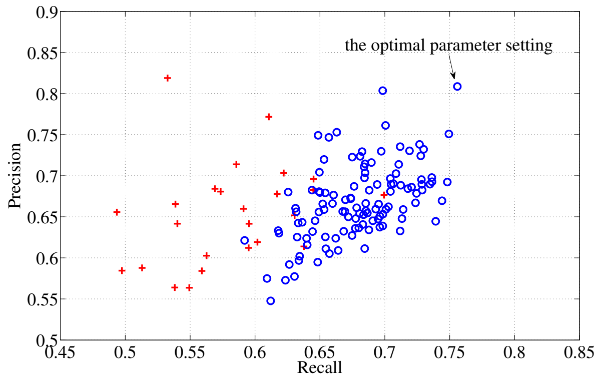


Figure 13: The precision-recall graph of different parameter settings for the `carsafe` and `baseline` schemes. A red cross (blue circle) indicates the precision and recall of the `baseline` (`carsafe`) scheme based on a particular set of parameters.

cameras, but a larger fraction of camera time is lost to the overhead of camera switching; and (2), longer switching times miss fewer events on average, but compromise dangerous event detection since the information from one camera is frequently stale (out-of-date). In comparison, we find the optimal `baseline` configuration is 12 seconds.

Switching Strategy Comparison. Figure 14 illustrates the F1 score when detecting each dangerous driving event (viz. DD, ID, CLC, LW, and TG) assuming both the `carsafe` and `baseline` schemes. For this experiment we use the optimal parameter configuration for both techniques, determined by our earlier experiment. On average `carsafe` has a 16% higher F1 score than `baseline`. One of the largest differences is when detecting CLC, which shows `carsafe` provides a 33% increase over `baseline`.

5.5 Multi-core Computation Planner Benchmarks

A set of experiments is conducted to estimate the average image processing rate of CarSafe. These experiments are carried out on a Samsung Galaxy S3 with an Android 4.2 OS.

Figure 15 presents CarSafe's processing time for both driver- and

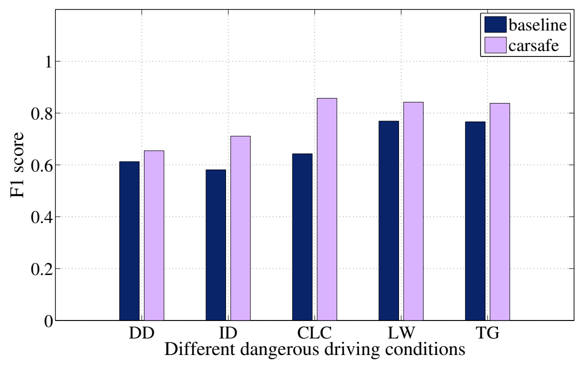


Figure 14: The F1 scores for detecting dangerous driving conditions using `baseline` (dark-color) and `carsafe` (bright-color) schemes.

road-based events. In CarSafe, multiple threads are dispatched in the multi-core computation planner component to boost the image processing performance. The frame rate of CarSafe image processing is boosted from 4.95 (running one thread) to 9.87 (running four concurrent threads) fps for the driver classification pipeline, and from 3.82 (running one thread) to 10.86 (running four concurrent threads) fps for the road classification pipeline. The standard deviation of the frame processing time is 7.88 (7.71) seconds for the driver (road) classification pipeline, which helps reduce the jitter in the frame processing latency. The figure shows that the processing time will not decrease when there are more than 4 threads concurrently processing. In order to boost the frame rate of image processing while maintaining an overall short waiting time, CarSafe concurrently maintains four threads within the multi-core computation planner.

6. RELATED WORK

Recognizing driving behavior (i.e., driver drowsiness, lane departure, and following distance) using fixed vehicle-mounted devices is an active area of research. In the case of drowsy driving, Bhowmick *et al.* [26] identifies driver drowsiness by detecting eyes and classifying eye states (viz. open and closed states) using an infra-red camera. By illuminating facial landmarks the nose and eyes are easily recognized, and by finding the eye shape the state of the eye can be classified. Vural *et al.* [42] predicts sleep episodes by monitoring facial actions (e.g., blinking, and yawning) using cameras and head movement – here the driver wears a head-mounted accelerometer. Aly [24] presents a real-time lane marker detector, which relies on a vehicle-mounted camera, bird's-eye view transformation and a gaussian filter. After transforming vehicle-mounted camera images to a bird's-eye view through inverse perspective transformation, the system filters the images via a 2D selective-orientation gaussian filter and thresholds the filtered images by zeroing all values below a set threshold. Finally, the system fits lines to the thresholded images, that identify lane markers, by using a RANSAC algorithm. Stein *et al.* [40] measures the following distance using a monocular camera mounted near the rear-view mirror. They design a vision-based Adaptive Cruise Control (ACC) system to determine the following distance by transforming the measured distances, i.e., the distances between the driving and preceding cars in the camera coordinate system, to real following distances in the world coordinate system based on a simple pinhole model. However, none of these examples of existing studies into driver behavior detection consider the limitations and challenges of a smartphone-based implementation. In developing CarSafe, we re-examine this family of techniques in light of real-world phone-in-car scenarios and investigate how they can be adapted to run on multi-core smartphones.

By integrating powerful sensors into top-end cars (e.g., cameras, radar, and ultrasonic sensors), manufacturers are bringing similar forms of driver behavior monitoring to the consumer market. These systems [15, 23, 6] warn the driver using acoustic (e.g., playing a voice recording [6]), tactile (e.g., seat and steering wheel vibration [23]) and visual (e.g., a blinking coffee-cup icon on the dashboard [15]) alerts when dangerous driving conditions are detected. Although the cost of safety technology is dropping, most safety technologies are not available in entry-level vehicles; it will be a decade before the vast majority of cars on the road today have these safety features built-in. In contrast, smartphone solutions such as CarSafe can be used in all cars (new or old) and represent a cheap and disruptive technology. Several commercial smartphone apps [11, 4, 13] are emerging. iOnRoad [11] and Augmented Driving [4] for example focus on monitoring the following distance using the back camera and LDWS [13] offers warning sounds when the vehicle departs a lane marker. To the best of our knowledge, none of these apps detect dangerous driving behavior using the front- and rear-facing cameras – they typically are only watching conditions on the road.

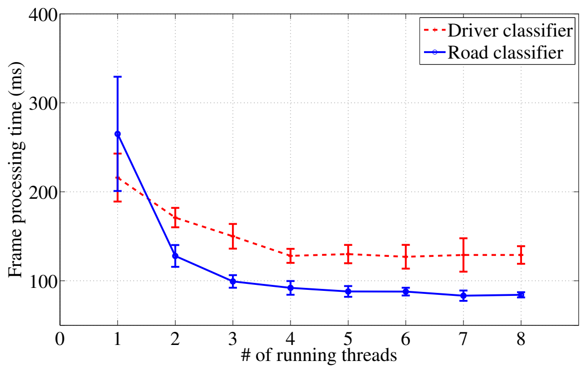


Figure 15: The frame processing time and time variance of the driver and road classifiers when replicating 1 ~ 8 running threads.

7. CONCLUSION

This paper describes the design, implementation and evaluation of CarSafe and presents results from a small-scale deployment of the app in the wild. The performance of the overall system looks very promising given the real challenges of different drivers, context and road conditions. Our future plans are to improve the current prototype with the goals of advancing the UI design (which was not a focus of this paper) in addition to gaining further improvements in the real-time performance of the CarSafe classification pipelines. We also plan to release CarSafe on Google Play [2] to get further input from the broader app user community. Finally, we plan to develop a set of open APIs enabling full access to the front and rear cameras and will start by studying more experimental open source platforms, such as, Tizen [22]. Our goal is to stimulate interest in dual camera sensing apps to encourage major platform vendors to solve this problem.

8. REFERENCES

[1] Active Shape Model. `http://code.google.com/p/asmlib-opencv/`.
[2] Android Apps on Google Play. `https://play.google.com/store`.
[3] Android Open Source Project. `http://source.android.com`.
[4] Augmented Driving. `https://itunes.apple.com/tw/app/augmented-driving/id366841514?mt=8`.
[5] BioID Face Database. `http://www.bioid.com/`.
[6] Bosch Debuts Low-Cost Drowsy Driver Alert System. `http://editorial.autos.msn.com/blogs/autosblogpost.aspx?post=caac1a6c-af18-43ae-8fda-244305f449ec`.
[7] Breakthrough Research on Real-World Driver Behavior Released. `http://www.nhtsa.gov/Driving+Safety/Distracted+Driving/Breakthrough+Research+on+Real-World+Driver+Behavior+Released`.

[8] Driver's Handbook - Lane change. http://www.mto.gov.on.ca/english/dandv/driver/handbook/section6.5.0.shtml.

[9] Face Detection for Windows Phone 7. http://facedetectwp7.codeplex.com/.

[10] FaceDetect - OpenCV. http://opencv.willowgarage.com/wiki/FaceDetection.

[11] iOnRoad. http://www.ionroad.com/.

[12] iOS 5 Face Detection with Core Image. http://www.bobmccune.com/2012/03/22/ios-5-face-detection-with-core-image/.

[13] LDWS 2.1. http://www.androidzoom.com/android_applications/tools/ldws-21_ljrs.html.

[14] Linux Media Infrastructure API. http://linuxtv.org/downloads/v4l-dvb-apis.

[15] No Doze: Mercedes E-Class Alerts Drowsy Drivers. http://www.autoweek.com/article/20081224/free/812249991.

[16] NYS DMV - Driver's Manual - Chapter 8: Defensive Driving. http://dmv.ny.gov/dmanual/chapter08-manual.htm.

[17] OMAP™ Processors - OMAP™ 4 Processors - OMAP4460. http://www.ti.com/product/omap4460.

[18] OsmAnd (OSM Automated Navigation Directions) - Navigation and Routing based on Open Street Maps for Android Devices. http://osmand.net/.

[19] Samsung Galaxy S4 - Captuers All The Fun. http://www.samsung.com/global/microsite/galaxys4/fun.html.

[20] Street View – Google Maps. www.google.com/streetview.

[21] The MUCT Face Database. http://www.milbo.org/muct/.

[22] Tizen: An Open Source, Standards-based Software Platform. https://tizen.org/.

[23] Volvo's Work within Active Safety. http://www.crp.pt/docs/A18S104-6_3_MariaFlink.pdf.

[24] M. Aly. Real Time Detection of Lane Markers in Urban Streets. In *IEEE IV '08*, pages 7–12, 2008.

[25] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[26] B. Bhowmick and K. Chidanand Kumar. Detection and Classification of Eye State in IR Camera for Driver Drowsiness Identification. In *IEEE ICSIPA '09*, pages 340–345, 2009.

[27] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[28] C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, June 1998.

[29] J. Canny. A Computational Approach to Edge Detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986.

[30] Y.-Y. Chiang and C. A. Knoblock. Automatic Extraction of Road Intersection Position, Connectivity, and Orientations from Raster Maps. In *ACM GIS '08*, pages 22:1–22:10, 2008.

[31] P. Finnegan and P. Green. The Time to Change Lanes: A Literature Review. Technical Report IVHS-TR-90-13, National Technical Information Service, Sep. 1990.

[32] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[33] Y. Freund, R. Schapire, and N. Abe. A Short Introduction to Boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(5):771–780, 1999.

[34] R. Knipling. PERCLOS, A Valid Psychophysiological Measure of Alertness as Assessed by Psychomotor Vigilance. Technical Report FHWA-MCRT-98-006, Federal Highway Administration, Office of Motor Carrier Research and Standards, October 1998.

[35] C. C. Liu, S. G. Hosking, and M. G. Lenne. Predicting Driver Drowsiness using Vehicle Measures: Recent Insights and Future Challenges. *Journal of Safety Research*, 40(4):239–245, 2009.

[36] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell. The Jigsaw Continuous Sensing Engine for Mobile Phone Applications. In *ACM Sensys '10*, pages 71–84, 2010.

[37] NHTSA's National Center for Statistics and Analysis. Distracted Driving 2010. Technical Report DOT-HS-811-650, National Highway Traffic Safety Administration, Sep. 2012.

[38] C. Papageorgiou and T. Poggio. A Trainable Object Detection System: Car Detection in Static Images. Technical Report 1673, October 1999.

[39] D. D. Salvucci and A. Liu. The Time Course of a Lane Change: Driver Control and Eye-movement Behavior. *Transportation Research Part F: Traffic Psychology and Behaviour*, 5(2):123–132, 2002.

[40] G. P. Stein, O. Mano, and A. Shashua. Vision-based ACC with a Single Camera: Bounds on Range and Range Rate Accuracy. In *IEEE IV '03*, pages 120–125, 2003.

[41] P. Viola and M. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. In *IEEE CVPR '01*, volume 1, pages I–511 – I–518 vol.1, 2001.

[42] E. Vural, M. Cetin, A. Ercil, G. Littlewort, M. Bartlett, and J. Movellan. Drowsy driver detection through facial movement analysis. In *HCI '07*, pages 6–18, 2007.

[43] T. Wang, G. Cardone, A. Corradi, L. Torresani, and A. T. Campbell. WalkSafe: A Pedestrian Safety App for Mobile Phone Users who Walk and Talk while Crossing Roads. In *ACM HotMobile '12*, pages 5:1–5:6, 2012.

[44] W. W. Wierwille, S. S. Wreggit, C. L. Kirn, L. A. Ellsworth, and R. J. Fairbanks. Research on Vehicle-Based Driver Status/Performance Monitoring: Development, Validation, and Refinement of Algorithms for Detection of Driver Drowsiness. Technical Report DOT-HS-808-247, National Highway Traffic Safety Administration, 1994.