Auto++: Detecting Cars Using Embedded Microphones in Real-Time

SUGANG LI, XIAORAN FAN, YANYONG ZHANG, WADE TRAPPE, JANNE LINDQVIST, and RICHARD E. HOWARD, WINLAB, Rutgers University

In this work, we propose a system that detects approaching cars for smartphone users. In addition to detecting the presence of a vehicle, it can also estimate the vehicle's driving direction, as well as count the number of cars around the user. We achieve these goals by processing the acoustic signal captured by microphones embedded in the user's mobile phone. The largest challenge we faced involved addressing the fact that vehicular noise is predominantly due to tire-road friction, and therefore lacked strong (frequency) formant or temporal structure. Additionally, outdoor environments have complex acoustic noise characteristics, which are made worse when the signal is captured by non-professional grade microphones embedded in smartphones. We address these challenges by monitoring a new feature: maximal frequency component that crosses a threshold. We extract this feature with a blurred edge detector. Through detailed experiments, we found our system to be robust across different vehicles and environmental conditions, and thereby support unsupervised car detection and counting. We evaluated our system using audio tracks recorded from seven different models of cars, including SUVs, medium-sized sedans, compact cars, and electric cars. We also tested our system with the user walking in various outdoor environments including parking lots, campus roads, residential areas, and shopping centers. Our results show that we can accurately and robustly detect cars with low CPU and memory requirements.

CCS Concepts: • **Human-centered computing → Ubiquitous and mobile computing systems and tools**;

ACM Reference format:
Sugang Li, Xiaoran Fan, Yanyong Zhang, Wade Trappe, Janne Lindqvist, and Richard E. Howard. 2017. Auto++: Detecting Cars Using Embedded Microphones in Real-Time. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 3, Article 70 (September 2017), 20 pages.
https://doi.org/10.1145/3130938

1 INTRODUCTION

Using smartphones to sense users' surroundings and learn about their contextual information has received much attention in the past few years [9, 12, 21, 24, 37]. A large number of such studies have been conducted for users when they are driving [3, 30, 35], while much less effort has been devoted to users when they are walking. In this study, we focus on sensing and learning an important context information for pedestrian users using their smartphones – whether there is an approaching car nearby.

The ability to detect oncoming cars using smartphones can enable/enhance several important ubiquitous applications. Firstly, with such information available, notifications could be sent out to alert distracted users (through sounds or vibrations) about approaching cars when they are about to enter an intersection or wander into roads (illustrated in Figure 1). Secondly, we could aggregate sensed cars in different areas to perform fine-grained traffic monitoring. Although navigation apps (*e.g.*, Google Map [2] and Apple Map [1]) have been widely adopted

Author's Address: WINLAB, Rutgers University, 671, U.S. 1, North Brunswik, New Jersey, 08902.

2474-9567/2017/9-ART70 $15
https://doi.org/10.1145/3130938

Fig. 1. Auto++ detects approaching cars using audio data recorded by smartphones. This capability can enable applications such as alerting distracted users (possibly through sounds or vibrations). Please note that the application logic such as alerting users is not part of Auto++.

to collect vehicle traces and provide real-time traffic information, some geographic areas (such as residential areas, college campuses, etc) may not have sufficient data because drivers in those areas do not turn on their navigation apps frequently enough. In such cases, pedestrian users are able to sense cars in their vicinity and provide additional traffic data that was not available earlier. With these additional data, we could better assess the traffic condition in our neighborhood, such as outside of the post office or a grocery store. Thirdly, the ability to detect incoming cars could also enhance emerging augmented reality (AR) applications/games by enriching their virtual maps – *e.g.*, marking a place as 'walkable' or 'accessible' on a Pokemon Go map. Finally, with the help of such sensing technique, smartphones can notify the users with hearing impairment via direct vibration or other alerting schemes on the connected wearable.

In fact, there has been investigation on detecting cars in other communities. For example, solutions based on Dedicated Short-Range Communications (DSRC) have been proposed and studied [36, 38], which utilize a special radio channel to establish bidirectional communications between a car and people who are walking nearby. Such solutions, however, require additional hardware modules to be included in both the vehicle and smartphones, which may not offer an immediately available solution. Computer vision based techniques have also been studied, such as those in [27, 34], which detects vehicles via images captured by smartphone cameras. The disadvantage of such solutions, however, stems from several factors, including the requirement of high CPU overhead that is needed for real-time vehicle recognition, as well as the requirement of inflexible camera facing direction, etc.

In this paper, we propose a system, referred to as Auto++, that accurately detects approaching vehicles by processing real-time audio stream directly captured by off-the-shelf smartphone microphones without any prior training or additional infrastructure support. Auto++ does not have the shortcomings of a radio based or vision based system; in addition, it offers several clear advantages: (1) it is self-contained and does not need any infrastructure support; (2) it applies completely unsupervised classification techniques and no prior training is needed for the system to operate; (3) it is accurate and has low false positive rate; and (4) it is robust as it works with different cars, roads, and different environmental noise. The main challenge we face is due to the characteristics of car sounds. Today's engines are becoming increasingly quiet, and the perceivable sounds are generated mainly by tire-road friction. This tire noise lacks both distinguishable temporal and frequency structures. As a result, many popular acoustic techniques such as Doppler Shift [8] or features such as MFCC [40] cannot be applied in our system. In this study, we establish a new feature that can discriminate between car

sounds and other environmental sounds. The feature is the maximal frequency component that crosses a power thread; when a car drives closer to the user, this frequency continuously goes up. Through edge detection, we can robustly extract this feature from the spectrogram of the audio signal.

The car sound's property does not only present challenges for car detection, but it also provides an opportunity for our detection algorithm to work cross different cars and roads – tire noise from different cars is likely similar to each other. Once the system detects a car, we can estimate its driving direction by applying cross-correlation function over audio streams from two microphones (many of today's smartphones have dual microphones). We can also cluster detections that are within a short time period to count the cars around the user.

We implement Auto++ over Android platform and collect 330 minutes of audio data from smartphone over the course of 14 months. Our evaluation involves seven vehicles whose types cover a broad range of cars that are seen on US highways, including SUVs, medium sedans, sports cars, compact cars, and electric cars. The audio is recorded by Auto++ in a range of different outdoor scenarios, from quiet parking lots, to noisy outdoor shopping center during business hours. Our results show that Auto++ is robust across different cars and various types of environmental noise. Specifically, we demonstrate that we can detect 91% of the cars even when they are more than four seconds away from the user. On average, our system can detect cars when they are 5.7 seconds away (while a naive scheme could only detect cars that are 2.4 seconds away). We note that Auto++ can also detect electric cars when they are 5.9 seconds away with 100% accuracy. Finally, we are able to estimate a car's driving direction with an average success rate of 84%.

2 MOTIVATION AND CHALLENGES

In this section, we first present the motivation and system assumptions. Then we discuss several challenges that arise due to car sound signal properties and limitations of the underlying hardware platform.

2.1 Motivation and System Assumptions

Auto++ performs the following three tasks in real-time: (1) car detection – whether there is a vehicle approaching the user, (2) car tracking – what is its direction, and (3) car counting – how many cars are passing or have passed the user within a given time window. It performs these tasks by analyzing the acoustic signal captured by built-in microphones. The ability to answer these questions can enable several new applications. For examples, we could alert the distracted pedestrian users (through sound, vibration, etc); we could collect fined-grained traffic data from pedestrian smartphones; we could enhance augmented reality tools or games; we could further notify the hearing-impaired users with the surrounding roadside context information.

In our system, we assume that the user carries a mobile device that is equipped with one or two microphones. Today, all smartphones and tablets have at least one microphone, and some have two microphones, such as Google Nexus 6 and Nexus 6P. Further, we assume that the microphones have a similar degree of sensitivity in all directions. Also, we assume that the smartphone is held in hand by the user, or in bag/pocket with connected headset. Finally, we assume that the smartphones have the orientation service available so that Auto++ can calculate the smartphone's relative direction with respect to the road.

2.2 Background on Acoustic Signal Processing

Sound Pressure Level (SPL) is the direct form of a sound signal measured by the microphone and describes the local pressure deviation caused by the sound wave.

Time Difference of Arrival (TDoA) [13] provides a measurement of the location of sound source. By calculating the Cross-correlation Function (CCF), Inter-channel Phase Difference (IPD), or Maximum Likelihood Function [6, 23, 39] of the signal observed at difference microphones, the possible location of the sound source can be estimated.

With three microphones, the sound source's exact position can be estimated via triangulation. Due to the limitation of today's smartphone, we could only use two microphones to infer the sound direction.

Short-Time Discrete Fourier Transform (STDFT) is used to divide a long signal into shorter segments of equal length and then compute the discrete Fourier Transform on each of the segments. STDFT is widely used in audio analysis [39] as it can reveal both temporal and spectral information of the signal at the same time. It is particularly useful in detecting and tracking a moving car, because we can observe how the frequency changes when a car is approaching. We provide details on how we could potentially adopt these features in our system, or, why we can't adopt them at all. Finally, we also provide a brief discussion on commonly used acoustic signal processing tools.

Mel-frequency Cepstrum Coefficients (MFCC) stems from speech processing, which is derived from STDFT. In MFCC, a signal is first transformed into frequency response via FFT, then mapped to Mel scale to compute the coefficients. Since our sound source is not speech, MFCC does not necessary give us more information for tire friction noise than an FFT.

Doppler Shift is the frequency change of a wave at the receiver when the source moves, which is often used in sound source localization [16, 20]. The use of Doppler shift requires that the signal has one or more frequency ranges on which energy concentrates. Car sound signal, however, does not exhibit such skewed energy distribution, and thus we can not use Doppler shift to detect or track a moving car.

2.3 Challenges in Designing Auto++

In designing Auto++, we face several significant challenges. When a car runs at a steady speed (without sudden acceleration), its sound consists mostly of tire noise instead of engine noise, which has very different traits from many other sound signals that have been studied. Furthermore, this problem is made even worse by the fact that we are capturing and processing the audio data using off-the-shelf smartphones that have hardware limitations.

Lack of Energy Concentration on Frequencies: Many sound signals have distinctive energy concentration on certain frequencies, such as those studied in [15, 39]. In such cases (illustrated in Figure 2(a)(b)(c)), we can focus on detecting energy on those frequencies to detect the presence of the sound source. To localize such sound sources, the angle of arrival can be easily computed via the inter-channel phase difference [39], or Doppler shift [8] can be applied to estimate the distance between the sound source and the device. However, the sound of a vehicle does not contain obvious energy concentration on any specific frequencies. For example, Figure 2 (d) depicts a scenario when a car is driving towards a user at the speed of 30 Km/h on a asphalt road. Its energy mainly concentrate in low frequency range, which is similar to the signal where there is no car (as shown in Figure 2 (e)), therefore calls for different techniques.

Lack of Temporal Structure: Several studies rely on the sound signal's temporal features to localize the sound source, such as the sound pressure level (SPL) peaks [19, 23], sound emitting times [23], etc. However, we do not observe obvious temporal features in our car sound signal due to its noise-like nature.

System Limitations: We run Auto++ on mobile devices, and the system design is thus limited by the available features on the hardware platform, e.g., how many microphones a device has, how far apart are these microphones, how sensitive are these microphones, what is the maximum sampling rate, etc.

In this study, we have carefully addressed these challenges, and our evaluation results in Section 6 show that Auto++ is accurate, timely, and robust.

3 AUTO++ DESIGN

In this section, we present the detailed design of Auto++, which consists of four main steps: pre-processing, feature extraction, car detection, and car direction estimation.

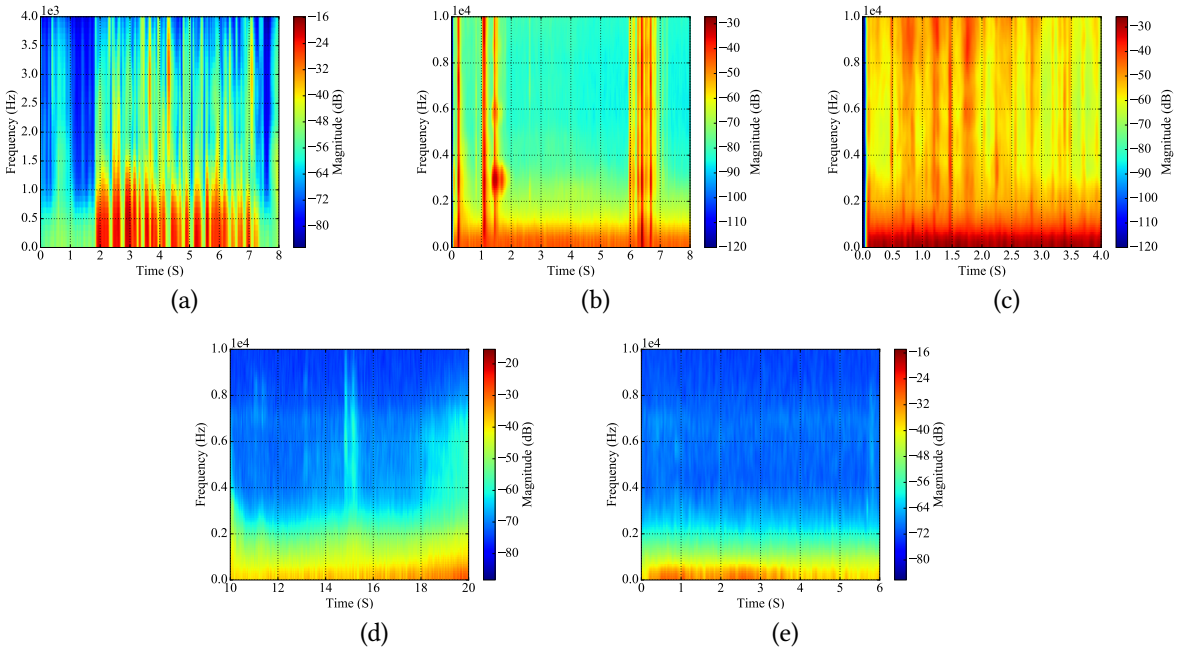


Fig. 2. (a) A typical speech signal has one or more frequency ranges on which acoustic energy is concentrated. (b) A spectrogram of door open and close sound. (c) A spectrogram of human steps. (d) A spectrogram of empty outdoor sound signal contains stable energy concentration in the low frequency range which is invariant in time domain. (e) When a car is driving towards a user at 30 Km/h on an asphalt road, its sound signal does not have clear energy concentration across different frequency ranges

3.1 Overview of Auto++

Fig. 3 depicts the overview of Auto++, which consists of the following components:

(1) **Pre-processing:** Before we can use the captured sound signal to detect, track, or count the cars, we first need to perform several pre-processing tasks, including segmenting the continuous stream of sound signal into smaller chunks, multiplying each chunk with a suitable window function, and conducting required processing on each chunk, such as discrete Fourier transform (DFT). These steps prepare the data for subsequent signal processing.

(2) **Feature Extraction:** Next, we extract features that can be used for car detection. Unfortunately, we find that those features that are commonly used to detect and localize sound sources are not suitable for our system (explained in Section 2) as car sound is dominated by tire noise and lacks distinctive spectral and temporal features. In this work, we have proposed new features that are unique to car sound signals and that facilitate accurate, timely, and unsupervised car detection.

(3) **Presence Detection:** Our presence detection algorithm is unsupervised. It can successfully discriminate between scenarios that involve an approaching car and those that do not involve a car (but with a high ambient noise), as early as possible.

(4) **Direction Estimation:** Our direction detection algorithm is based on Time Difference of Arrival (TDoA) of the sound signal. By computing the cross-correlation function over signals from two channels, we can provide a coarse direction estimation of the sound source.

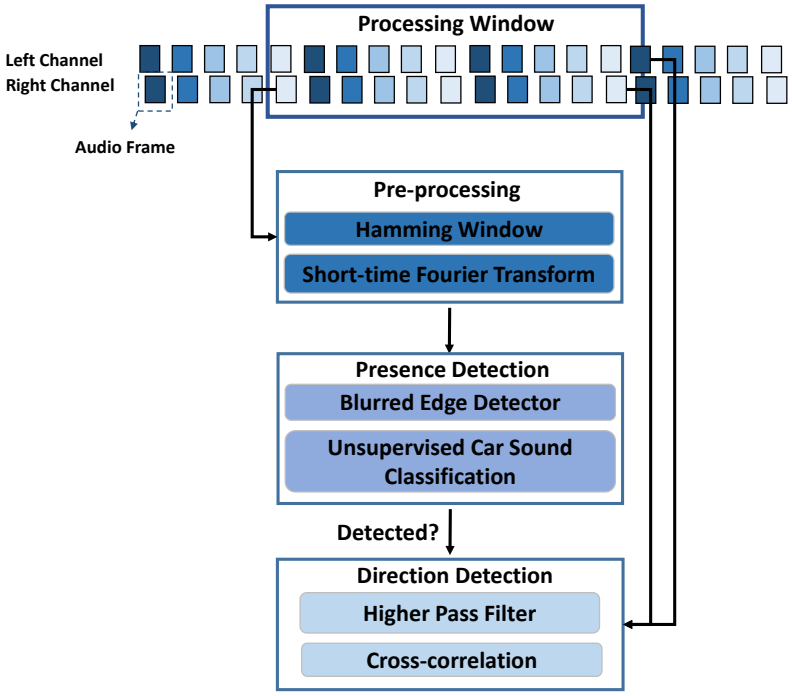


Fig. 3. The Auto++ system involves the following four components: (1) signal pre-processing, (2) feature extraction, (3) presence detection, and (4) driving direction estimation. Auto++ does not require any prior training for any of these steps.

3.2 Sound Signal Pre-Processing

Firstly, we segment the continuous stream of audio data into chunks of equal duration, each chunk referred to as a processing window. In our system, a typical window length is .5 second, which means we run our car detection algorithm every .5 second. Secondly, our detection algorithm involves the calculation of the short-time Discrete Fourier Transform (STDFT) of the audio samples in a processing window. For this purpose, we further divide a processing window into slots of equal length, and multiply the hamming window function with each slot. We then perform the Fourier transform over each slot to obtain the STDFT of the processing window. The slot duration determines the temporal granularity of our STDFT calculation.

3.3 Auto++ Feature Extraction

Next, we propose our car detection feature that addresses the unique challenges of car sound signals, and our feature extraction method that facilitates accurate, timely, and unsupervised car presence detection.

3.3.1 Top-Right Frequency (TRF): Maximum Frequency Whose Power Reaches a Certain Threshold. We discover our feature after carefully examining the car sound signal's STDFT results. Fig. 4(a) plots the STDFT result of the sound signal from a 2007 Toyota Camry during its drive of 100 meters. We started collecting audio samples when the car was 100 meters away, and ended recording when it drove past the user. The recording was done using a Nexus 6 smartphone. From the figure, we observe that when the car gets closer to the receiver, the energy across all frequency components rapidly increases, forming a mountain-like shape. The energy reaches the maximum when the car passes by the user's location.

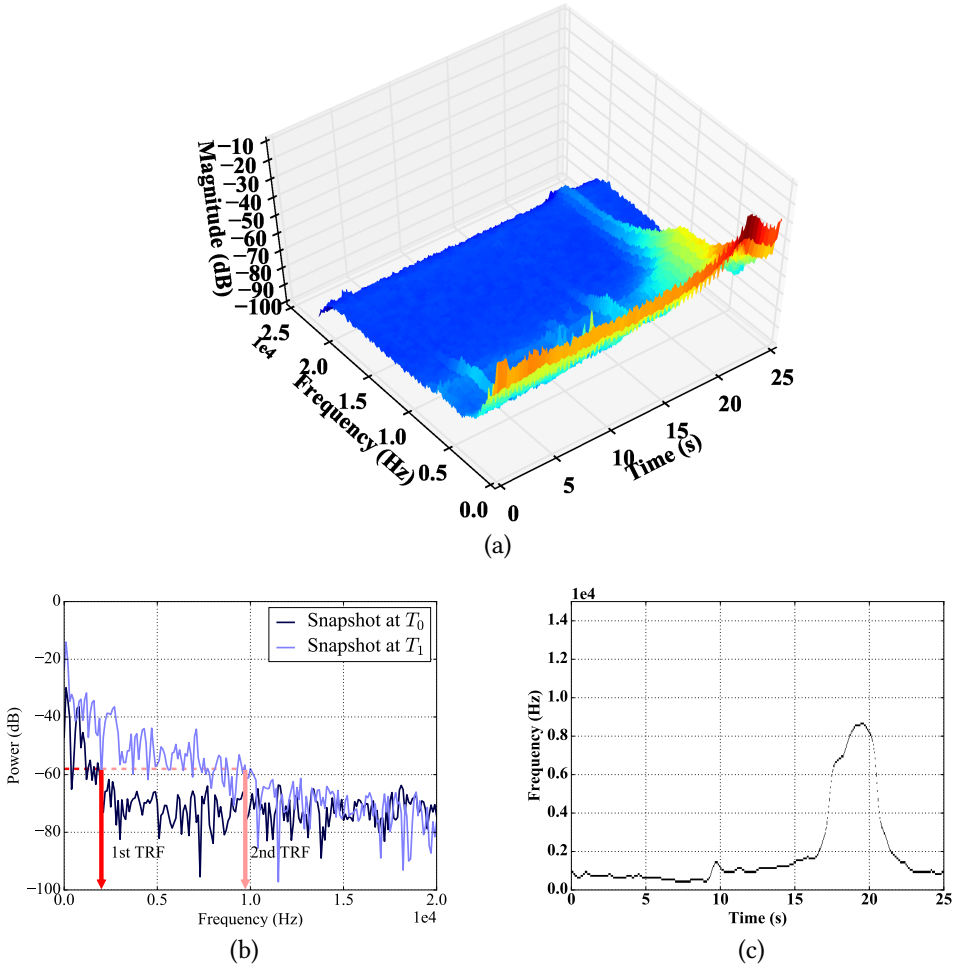


Fig. 4. (a) The STDFT result of the sound signal recorded when a 2007 Toyota Camry drove from 100 meters away from the user until a few meters past the user. (b) Two STDFT snapshots taken at different timestamps, in which we observe higher TRF when the car gets closer. (c) The TRF trace extracted by the blurred edge detection algorithm.

Let us look at two STDFT snapshots in Fig. 4(b). For each STDFT snapshot, we observe that the signal power drops as we go from lower frequency components to higher frequency components. This is because higher frequency components experience higher path loss. Let us first choose a particular power threshold value, *e.g.*, $-58dB$, and then find the corresponding highest frequency component whose power level is at or above this threshold. We refer to this frequency value as the *Top-Right* frequency, or *TRF* in short, because if we draw a rectangle whose height is given by the power threshold value (rectangle illustrated in red in Fig 4 (b)), the frequency of the top right corner of this rectangle is the frequency we are looking for. When the car gets closer to the user, the signal's TRF value becomes higher. In general, TRF can be calculated as:

$$TRF(n) = \max\left\{\underset{f}{\text{argmax}}(S(n,f))\right\},$$

where $S(n, f)$ is the power of the signal at frequency f and time window n. The term $S(n, f)$ is less than the power threshold T, and TRF(n) is the maximum f among all the frequencies whose $S(n, f)$ reaches the power level T. We calculate the TRF value for each slot to form a TRF trace, and *use the TRF trace as the feature for subsequent car detection.*

3.3.2 Blurred Edge Detection Based Feature Extraction. A straightforward way of generating a TRF trace is to adopt a fixed power threshold across all slots. Implementing this method in practice, however, is very challenging because it is hard to choose a suitable power threshold that works across cars and road conditions. Furthermore, adopting a single power threshold is not robust in real environments; in an outdoor environment, activities in the environment (*e.g.* bird singing, footsteps, etc.) can easily lead to high frequencies whose power exceeds the power threshold. As a result, the resulting TRF trace will not be smooth, sometimes even discontinuous.

In order to address the challenges associated with the simple threshold-based TRF trace extraction, we next develop an edge detection based method to extract TRF values from the STDFT result robustly and efficiently. As shown in Figure 4 (a), when driving closer to the user, the car generates sound signal that exhibits a continuous, blurred slope. The edge of this slope is the TRF trace of the car. We then adopt a Blurred Edge Detector (BED) to extract this edge, which is more robust than the power threshold based approach.

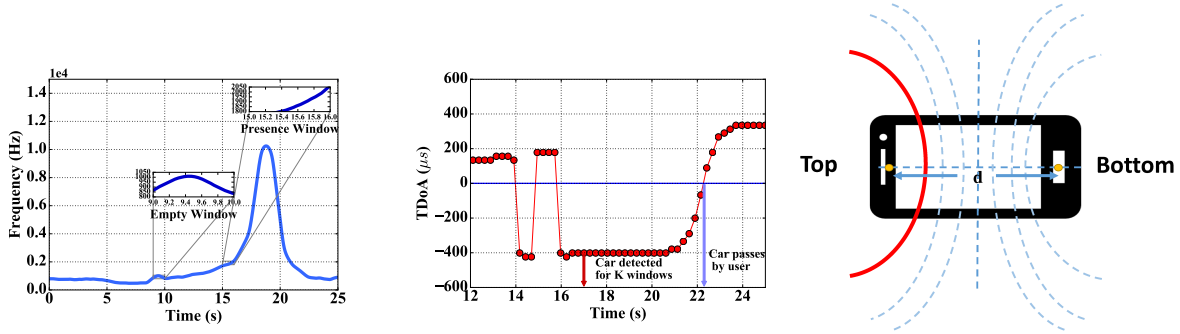
Our BED-based TRF extraction approach relies on both spectral and temporal information to identify the intensity gradients of the spectrogram. These gradients contain the desired TRF trace. Specifically, we adopt a Canny edge detector [5], a widely used technique, in Auto++. It includes the following steps: (1) We suppress those frequency components whose power is below a threshold by setting them to -100 dB. (2) We apply a Gaussian filter to smooth out noisy dots and lines. (3) We calculate the intensity gradient of the spectrogram to obtain the outline of the area that has a consistent power increase. (4) We apply non-maximum suppression to thin the area into edges. (5) We track the main edges by removing all the edges that are weak and not connected to strong edges. After finding the edge (shown in Fig. 4(c)), we fill the missing points on the edge with their previous neighbours' values to form a consecutive series. We call this series a *TRF trace*. We conduct such edge detection at the granularity of a *processing window*. Each processing window consists of multiple slots; we detect the edge within each window and uses the moving average over 200 slots to smooth the results.

3.4 Unsupervised Car Presence Detection and Counting

Next, we explain our car detection scheme, which is light-weight and unsupervised. Our car presence detection scheme includes two steps– processing window classification and event detection. In the window classification phase, we classify the current processing window (that contains all the audio samples recorded in the last .5 second) as either 'empty' (meaning no car is detected within this window) or 'car present' (meaning at least one car is detected within this window). Our window classification is based upon whether the TRF values within the window consistently increase. If a large fraction of TRF values are greater than or equal to their immediately preceding TRF value, then we label the window as a car-present window. This approach ensures only the continuously increasing portion of a TRF trace is counted as a car presence event, eliminating the peaks caused by ambient noise and other activities. For example, Fig. 5 (a) shows the TRF values in a typical empty window with a high noise level, as well as the TRF values in a window that contains an approaching car.

In some cases, it is important to find out how many cars are around a user. In order to get the car count, we merge consecutive car-present windows within a short time interval to avoid counting the same car multiple times.

Finally, we note that our detection scheme can work with other features. In our evaluation (Section 6), we also used sound pressure level (SPL) in our detection algorithm.



(a) (b) (c)

Fig. 5. (a) TRF values in a car-present window often exhibit different patterns from those of an empty window that has short-term ambient noise. In the former case, the TRF values often continuously increase while in the latter case, the TRF values first increase and then decrease. (b) The TDoA values when the car gets closer to the user. The red arrow marks the time when the car has been detected in K consecutive windows, while the blue arrow marks the time when the car passes the smartphone. (c) Example hyperbolas we have for direction estimation. The red solid hyperbola on the left plane indicates that a car is driving from the left side.

3.5 Unsupervised Car Direction Estimation

After detecting an approaching car, we estimate the car's driving direction. We attempt to do so by measuring the time difference of arrivals (TDoA) between sound signals captured by the microphones (if multiple are available on the mobile device). Traditionally, TDoA has been used to provide precise location information for the sound source [8]. To calculate signal TDoA, we apply the cross-correlation function (CCF) on these two signals $f(\tau)$ and $g(\tau)$ to calculate the lag between them:

$$(f \star g)[\tau] = \sum_{t=-\infty}^{\infty} f^*[t]g[t+\tau], \tag{2}$$

where f^* denotes the complex conjugate of f and τ is the delay. The corresponding t value of the highest peak is the estimated delay between the two channels.

However, when applying the above method on smartphones in our problem, we face the following constraints:

1) *Unstable SPL measurements*: TDoA calculation is based on the SPL values from both channels. However, as SPL itself can be easily affected by ambient noise especially when the sound source is far away from the receiver (as in our case), the cross-correlation results of the two channel is usually not accurate enough.

2) *Limited Number of Microphones*: Precise angle calculation requires at least three microphones – we can triangulate using the three sets of TDoA values. However, most of the off-the-shelf smartphones are equipped with one or two microphones. With two embedded microphones, we can only calculate one set of TDoA values.

3) *Low Sampling Rate*: The TDoA resolution is highly related to the sampling rate of the recording device. The smartphone's sampling rate is not higher than $44Khz$, which is not sufficient for high TDoA resolution.

4) *Tiny Distance between Two Microphones:* The distance d (as shown in Fig. 5 (c)) between two microphones determines the granularity of the distinguishable TDoA as well as the effective detection distance. On smartphones, the distance between microphones is usually too small to provide sufficient resolution.

In order to address the challenge of unstable SPL measurements, we propose to start TDoA calculation only when we have detected the same car for K consecutive windows. At this point, the SPL measurements are more stable, leading to more accurate TDoA results and direction estimation.

Fig. 5 (b) shows how the TDoA value varies as the car approaches while the user remains stationary on the sidewalk. As soon as a car has been detected in 4 consecutive windows at the 17th second (marked by the red arrow), the TDoA value becomes much more stable than before. It remains stable until the 21st second. In this example, the car passes the user at the 22nd second (marked by the blue arrow). Even when the TDoA value stabilizes, its absolute value may not be directly used to localize the car, as it is limited by many geometric parameters such as the angle and the distance between the smartphone and the road. Instead, we only consider the sign of the first valid TDoA value and estimate from which direction the car is driving based on the sign. Fig. 5 (c) illustrates several possible hyperbolas determined by the measured TDoA values. When we have a negative TDoA value, it indicates the sound source is located at a hyperbola on the left side (marked by red). That is, a car is coming from the smartphone's top side.

Finally, we note that, when applications require high TDoA resolution and direction estimation, we could address the hardware-related challenges 2), 3) and 4) by connecting external microphones with higher specs to smartphones.

4 SYSTEM'S TIMELINESS, ACCURACY AND USABILITY

For a sensing system such as Auto++, it is quite natural to raise the concern of (1)having late event detection, (2) having inaccurate event detection, and (2) having excessive notifications or updates even when the detection is timely and accurate. While the first two concerns are closely related to system performance, the third concern is focusing on its usability. Here, we would like to discuss how we can address these concerns to adopt different application requirements.

In general, we categorize the potential applications into two classes – instant sensing and daily logging. In the first class, the main objective of these applications is to detect the presence of a car whenever it is approaching users. As a result, the timeliness requirement is critical, *i.e.*, we want to detect the car as early as possible so that the user has enough time to react. However, when a system is able to capture the car sound when it is far away, it may encounter higher false positive rate as it is sensitive to signal with low Signal-to-Noise Ratio (SNR). In order to understand this trade-off between detection interval and false positive rate, we have conducted experiments by varying the processing window size (see Section 6.1). On the other hand, the second class applications are focusing on continuously logging the presence of cars which is in the vicinity of a user to provide additional traffic information. These applications do not require the detection interval as long as the previous ones, hence we can set the system parameters towards lower false positive rate.

For applications in the instant sensing class, even the detection is in time and accurate, however, it is still bothersome if there are excessive notifications. In order to address the usability of the system, we argue that Auto++ should decouple sensing and event notification/updating, and that the notification/updating module should be made tunable based on the exact context the user is in.

The event notification/update frequency may need to vary, when we use different applications in different situations –*e.g.*, a user might prefer less frequent alerts about incoming cars when walking in a crowded downtown area during peak hours, compared to walking on a suburban street during off-peak time; games such as Pokemon Go might need more frequent updating than health applications such as Fitbit.

With this in mind, we design Auto++ such that it dynamically adjusts the notification/updating frequency as desired. First, by giving users the ability to configure their preferred setting, we could trigger Auto++ only in specified situations - e.g., when the user is about to enter an intersection or walking on the sidewalk in a less crowded area. Leveraging existing technologies such as geo-fencing [29], the system could detect whether the user is in such a situation and then activate/deactivate car detection accordingly.

In addition to user-specified settings, Auto++ can also filter out unnecessary event notification/updating through detailed context sensing. For example, the report will only be sent out if an approaching car is detected

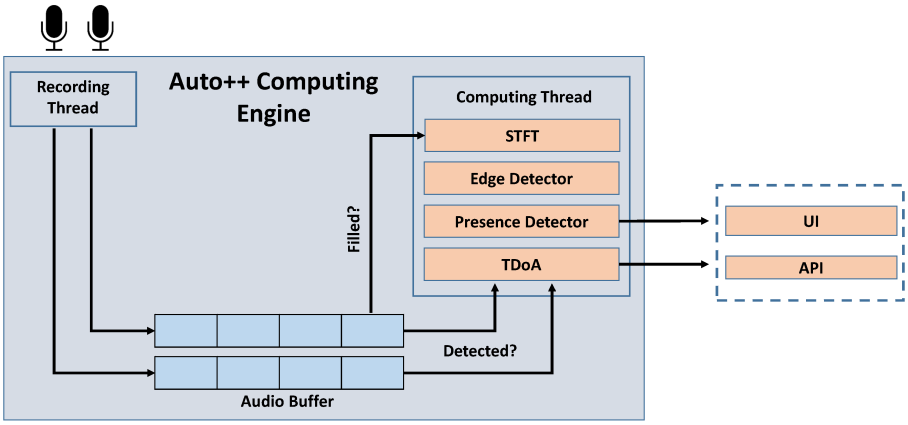


Fig. 6. Implementation of Auto++ Computing Engine on Android platform

while the user is walking towards/along the same road. Such sensing can be achieved by using built-in inertial sensors, GPS, etc.

5 ANDROID IMPLEMENTATION OF AUTO++

We have implemented Auto++ using the Java and OpenCV native library on Android platform. The raw audio is recorded at a 44 KHz frequency, with a 16 bit pulse-code modulation (PCM). To achieve real-time processing and event updating, we have optimized the code base of Auto++ and adopted concurrent threads to avoid stalling either sensing/recording or signal processing. As shown in Fig 6, the Recording thread handles data collection from microphones and populates the processing buffer, while the Computing thread processes the buffered data and sends the results (car detection + direction estimation) to a user interface or applications (through API).

Memory and CPU Usage Profiling: We have tested Auto++ on both Nexus 6 and Nexus 6P. The core processing engine of Auto++ runs on-demand, *i.e.*, it is only triggered when specific contexts such as walking on streets are sensed through GPS and inertial sensors (as discussed in Section 4). As a result, we consider the processing engine is in one of the three stages: idle, presence detection, and direction estimation.

The processing engines stays idle most of the time, and enters the car detection stage when the user-specified context is detected. In the presence detection stage, the engine performs sound recording, STFT calculation, edge detection, presence detection. When Auto++ detects a car-present window, it enters the direction estimation stage. In the direction estimation stage, the system needs to first detect the car's presence in $K-1$ consecutive windows, and then perform TDoA calculation via cross-correlation function.

Table 1. Auto++'s memory and CPU usage profiling. Among the three stages, direction estimation consumes the most CPU cycles, but still very low, 3.2% on Nexus 6 and 1.8% on Nexus 6P. The memory consumption remains within the pre-assigned memory range during the execution.

	Nexus 6 (Idling)	**Nexus 6 (Presence Detection)**	**Nexus 6 (Direction Estimation)**	**Nexus 6P (Idling)**	**Nexus 6P (Presence Detection)**	**Nexus 6P (Direction Estimation)**
Memory (MB)	36.4	36.4	36.4	32.8	32.8	32.8
CPU (%)	0	2	3.2	0	1.1	1.8

Table 1 summarizes the memory and CPU usage profiling of Auto++ on the two phones. Memory allocation is dynamically done by Android OS when the app boots up. During the execution of Auto++, its total memory consumption remains within the pre-assigned range, which is 36.4 MB on Nexus 6 and 32.8 MB on Nexus 6P. Direction estimation consumes the most CPU cycles than the other two stages, and its CPU utilization is only 3.2% for Nexus 6 and 1.8% for Nexus 6P. As a result, we believe our Auto++ computing engine is lightweight enough to become one of the default background services on mobile platforms.

6 EVALUATION

Our experiments are aimed at evaluating the two key functionality for Auto++: (1) detecting the approaching car on a road when the user is walking towards the road or on the sidewalk, and (2) estimating the direction from which the car is approaching. We have conducted extensive data collection to evaluate Auto++'s performance in these cases. Our results show that Auto++ can reliably detect a vehicle's presence 5.7 seconds in advance on a parking lot. On more crowded roads, Auto++ can count the approaching cars with an average error of .6 cars per minute, when the actual number of passing cars is no more than 6 per minute. Auto++ can estimate an approaching car's direction with 84.3% probability; its accuracy increases to 93.3% when multiple cars are driving in the same direction.

6.1 Accurate and Early Car Detection

Setting I: Stationary User & Single Car: We first evaluated how well Auto++ can detect approaching cars on an outdoor parking lot, where the user stood still on the sidewalk while the car drove closer. We conducted the following experiments in a quiet parking lot to collect audio data when a car was approaching the user. Cars were driving on a new asphalt road, which is a common road type in the US and generally more quiet when it is new [14]. We started an experiment by having the emulated user wave to the driver, who then began to drive the car from 150 meters away towards the user's direction at speed of 30Km/h. In this set of experiments, we used a Google Nexus 6, which was fixed on a tripod with one meter distance to the road, to record the sound via both microphones. To reduce the impact of wind, we mounted windscreens on the microphones. We ended an experiment after the car drove 20 meters past the user. Throughout the entire experiment, we controlled the driving speed to be constant and under the required speed limit. We repeated this experiment to collect data over 7 vehicle models which covered a large fraction of vehicles operated in North America, including electric cars, sedans, sports cars, and sport utility vehicles.

In total, we collected 210 audio tracks that contain an approaching car using the above method, each audio track 20 seconds long. We evaluated the two proposed detection algorithms: TRF, which performs feature extraction based upon a fixed power threshold, and BED-TRF, which performs feature extraction using blurred edge detection. We compared their performance with a baseline sound pressure level (SPL) based detection algorithm – classifying an audio track as 'containing a car' when its SPL is above a threshold. Considering that SPL is highly sensitive to ambient noise, we applied a high-pass filter with the cutoff frequency of 1000 Hz to reduce the impact of noise.

In the evaluation, our objective is to report whether the detection algorithm can accurately detect whether a car is present in an audio track, and how early can it detect the car. The metric we use here is *detection interval*, the interval between the time when a car is first detected and the time when the car is expected to pass the user's location. The larger the detection interval is, the earlier we could detect the car. We present the detection interval breakdowns in Table 2. Among the three algorithms, BED-TRF fares the best, while SPL the worst. Even when the car is 4 seconds way, BED-TRF can detect its presence with 91% probability. When the car is 2 seconds away, BED-TRF and TRF can detect car presence with the 100% and 97% probability, respectively. With the data sets we have, we observe the false positive rates are 0 for all algorithms. This shows that BED-TRF has a reliable detection performance.

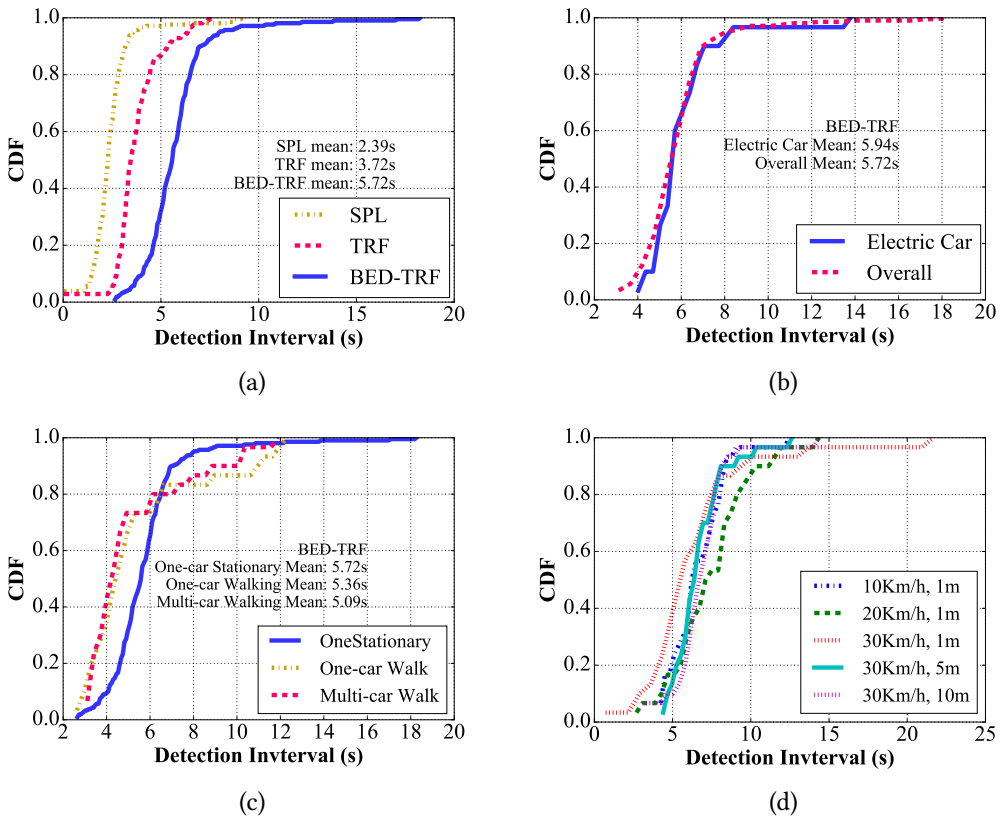


Fig. 7. (a) CDFs of the three algorithms' detection intervals show that BED-TRF provides the earliest detection; (b) CDF of the electric car (Volt)'s detection intervals shows that Auto++ can detect electric car as well as other cars (BED-TRF used here); (c) Average detection interval becomes shorter when the user is walking, but an average detection interval of 5.36 second is still very timely. When there are multiple cars, the average detection interval is only degraded by 6%; (d) CDFs of the detection intervals are similar when we vary the driving speed and the distance between the road and the phone.

Fig. 7(a) depicts the Cumulative Distribution Function (CDF) of the detection intervals of all three algorithms. The average detection interval of SPL, TRF, and BED-TRF is 2.4, 3.7 and 5.7 seconds, respectively, with BED-TRF significantly outperforming the other two. BED-TRF has the best performance because its feature extraction is the most robust. In the rest of the evaluation, we thus focus on BED-TRF.

Finally, Fig. 7(b) shows even *an electric car can be accurately and timely detected* by Auto++. We find that the mean detection interval for the electric car (Chevrolet Volt) is 5.9 seconds, which is even slightly better than the overall mean detection interval. This result is very encouraging as there is a fear that electric cars are too quiet, and thus dangerous to the pedestrian.

Setting II: Mobile User & Multiple Cars: Next, we evaluated how well Auto++ can detect approaching cars while the user is walking on the sidewalk of a parking lot (smartphone in his hand), with one or more cars coming from the same direction. When we had multiple cars, we made sure that they follow the preceding car after a safe distance (more than 2 seconds away). In this setting, we collected 60 audio tracks in total.

Compared to the first setting, we varied two parameters in this setting: user activity (standing still vs. walking), and number of approaching cars (one vs. two). We report the CDF of the detection intervals in Fig. 7(c). We observe that, user activity has larger impact on the average detection interval – a walking user's average detection interval is 5.4 seconds while a standing user's detection interval is 6.8 seconds, but we are still able to detect the presence with 90% of probability when a car is 3 seconds away. The number of cars has a much smaller impact. For a walking user, when the number of approaching cars changes from 1 to 2, the average detection interval changes from 5.4 seconds to 5.1 seconds, and we can detect a car's presence with 93.3% of probability when a car is 3 seconds away.

Setting III: Noisy Environments: In the third set of experiments, we investigated whether our presence detection algorithm is robust enough to detect cars in rather noisy environments. Towards this goal, we recorded the ambient sound in several typical environments frequented by pedestrians when there is no running car nearby, including shopping centers, campus roads, and residential areas during busy hours on a weekday. Typical sound sources of the ambient noise in these scenarios include steps, talking, door open/close activities, car sound from a far distance, etc. We recorded the sound in each environment for a total length of 10 minutes, consisting of 30 20-second clips. We overlaid these noise clips over those that contain an approaching car (which we collected in Setting I), emulating a 'car-running-in-a-noisy-environment' scenario.

Table 3 summarizes BED-TRF's true positive rate and false positive rate in the original quiet environment (parking lot in Setting I) as well as three noisy environments. We observe that, in noisy environments, the detection performance does degrade, but the results are still quite encouraging. For example, on campus roads with students walking/chatting and cars/school buses running, we can detect approaching cars with an average true positive rate of 97.2%, an average false positive rate of 3.3%, and an average detection interval of 4.2 seconds for true positive events. As expected, the results at the shopping center are worse than those on a campus road and in a residential area, because there are many more activities there that generate ambient noise, such as people going in/out of stores, cars running in the shopping center parking lot, etc. In our future work, we will try to improve the detection performance at a shopping center by learning and recognizing noise generated by these different activities.

Table 2. Detection interval distribution breakdown. BED-TRF can detect cars when they are 4 seconds aways with 91% of probability

detection interval	SPL	TRF	BED-TRF
>0.5s	0.96	0.97	1
>2s	0.71	0.97	1
>4s	0.03	0.3	0.91
>6s	0.02	0.07	0.46
>8s	0.02	0.25	0.2

Table 3. Auto++'s detection accuracy at campus roads and residential areas is only slightly worse than the performance on quiet parking lots. We see more degradation in detection accuracy when we use Auto++ at a shopping center because there are a multitude of activities that generate strong ambient noise.

	Quiet Parking Lot	Campus Road	Residential Area	Shopping Center
Detection Interval (s)	6.8	4.2	3.6	3.2
TPR (%)	100	97.2	94.8	83.8
FPR (%)	0	3.3	13.3	13.3

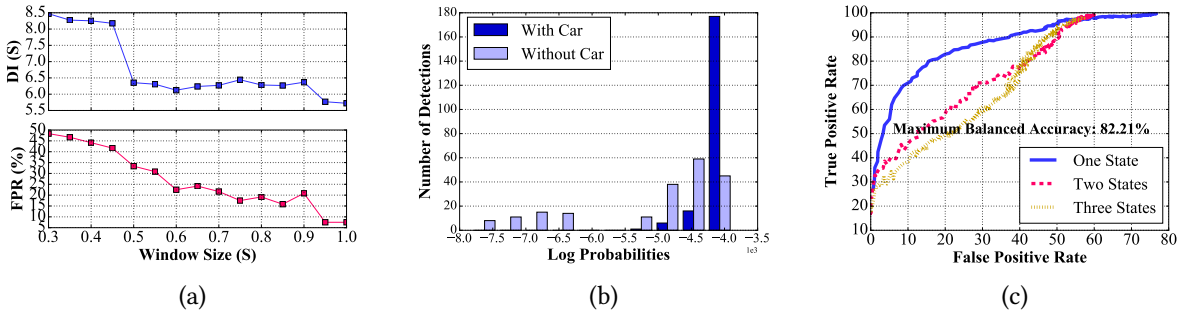


Fig. 8. (a) Impact of window size on detection interval and false positive rate. Longer window size leads to shorter detection interval but lower false positive rate. (b) An example of log probabilities distribution of HMMGE test indicates that we can separate audio segments with cars and audio segments without cars via setting a proper log probability threshold. (c) We find the maximum balanced accuracy is 82% when HMMGE only contains one state

Setting IV: Varying Driving Speed and Distance between Road and Phone: In the fourth set of experiments, we study whether the performance of Auto++ is robust against parameters such as how fast the car is driving towards the user, and how far the user/microphone is close to the road. For this purpose, we first had the user stand one meter away from the road, and drove the car at a speed of 10 Km/h, 20 Km/h, and 30 Km/h. Then we fixed the driving speed at 30 Km/h, while varying the distance to be 5 and 10 meters away from the road.

Figure 7 (d) shows the distribution of detection intervals for these settings. The results show that the two parameters do not have a significant bearing on the detection interval. In fact, according to Iwao et al. [17], a slower car in general generates lower tire friction noise, and we may detect it at a shorter distance away compared to a faster driving car. However, a shorter distance does not necessarily lead to a shorter detection interval as its speed is also lower. Finally, we note that the distance between the phone and the road does not have a noticeable impact on the detection interval because it is in general much shorter than the distance between the phone and the car anyway.

Setting V: Varying Processing Window Size: In this set of experiments, we investigate the impact of processing window size. Specifically, we perform the BED-TRF algorithm over audio files containing car sound (see Setting I for details) to evaluate the detection interval when we have different window sizes, and perform the algorithm over audio files without car sound (see Setting II for details) to report the false positive rate with different window sizes. For both evaluations, we vary the processing window size from 0.3 seconds to 1 second with 0.05 seconds increment.

We report these two results in Figure 8 (a). We observe that a larger processing window size leads to shorter detection intervals when there is actually a car, but meanwhile lower false positive rates in when there is no approaching car. The reason is that our presence detection algorithm observes the percentage of increasing TRF values within a window. Given a fixed percentage threshold, a larger processing window size inevitably results in a later detection, but it can also help filter out 'false' energy peaks caused by various environmental noise.

Setting VI: Comparison with MFCC-based Detection: In this set of experiments, we aim at comparing our approach with an MFCC-based machine learning approach that is commonly used in acoustic signal processing. We compare their detection accuracy here. For this MFCC based scheme, we implement Hidden Markov Model with Gaussian Emissions (HMMGE) [25] to discriminate whether an audio segment contains any car sound. In HMMGE, given model parameters and observed data (test audio segment), we are able to calculate the likelihood

of the data belonging to the model. We can then make a decision to accept or reject this data by comparing the likelihood against a carefully chosen threshold.

We first calculate MFCC for each audio segment. As mentioned in Section 2.3, there are no noticeable spectral and temporal features for tire friction noise, and we therefore focus on those audio segments in which the car sound is more pronounced. Specifically, we choose the last-four-second audio clips before a car passes the user, which are collected in a quiet parking lot when cars were driving at 30Km/h (see Setting I for details). The rationale is that the MFCC features for these audio samples are more likely to detect the presence of the car. We divide each of these audio segments into four one-second chunks and calculate their MFCC. On the other hand, we also calculate MFCCs for audio chunks (one-second length) which do not contain car sound and which are collected on quite parking lots, campus roads, residential areas, and outdoor shopping centers (see Setting III for details).

We train the MFCC HMMGE model using the data containing car sound and test both the true cases (with test data containing car sound) and false cases (with test data that do not contain car sound). We perform 50 iterations of random sub-sampling validation; in each iteration, we randomly choose 75% of the true data set as the training data, and 25% of both true data set and false data set as the test data. In order to explain the results more clearly, we randomly pick an iteration, and show the histograms of the log probabilities of detection for true test data and false test data in Figure 8 (b). This plot shows that these two histograms are distinguishable by using a suitable log probability threshold.

Next, we compute the true positive rates and false positive rates over 100 log probability threshold values, varying from -4000 to -5000 with an increment of -10. We also consider three variations of the HMMGE model, with the number of HMM states being 1, 2, and 3, respectively. Figure 8 (c) depicts three ROC curves [22], one for each model variation. The results show that the one state model fares the best among all three variations.

In order to compare the MFCC-based approach with our approach, we use balanced accuracy, $\frac{TPR \times N_T + (1-FPR) \times N_F}{N_T + N_F}$, where N_T and N_F are the number of true tests and the number of false tests, respectively. HMMGE can achieve at most 82% balanced accuracy, even when we only consider the last 4 second audio samples (car detection from these samples is much easier than that from other samples), which is still much lower than the balanced accuracy of 97% provided by our BED-TRF approach. This is because our approach is capable of capturing the unique features in both signal amplitude and frequency when a car is approaching, while MFCC is not a very useful feature in such cases.

Setting VII: Counting cars by Mobile User in Crowded Areas: Finally, we investigate whether Auto++ can detect and count approaching cars in areas with a relatively high traffic volume, while the user walking around in the area (on sidewalks) with the smartphone in hand. In this set of experiments, we collected data on a campus main road, where cars were not driven by our participants, but by regular drivers. In total, we recorded 180 minutes of audio data, with each audio track 1 minute long. We extracted the ground truth (car passing the user and the corresponding timestamps) by locating the TRF peaks in each trace offline. We then compare the number of cars detected by Auto++ with the ground truth, and call the absolute difference of the two as *counting error*. If two cars are very close to each other (less than 2 seconds away from each other), we consider them as one car.

We report the counting error histogram with respect to different traffic volumes in Fig. 9 (d). The results show that the average counting error is 0.58 car per minute when the user is standing still, and 1.04 car per minute when the user is walking, when the actual traffic volume is 5 or 6 cars per minute. When the actual car number per minute is greater than 8, the counting error is 0.91 (standing) and 2.02 (walking) per minute. This result indicates that Auto++ can accurately detect and count approaching cars when the traffic volume is moderate. As a result, we believe that Auto++ offers a viable approach for fine-grained traffic monitoring when existing traffic monitoring tools are insufficient.

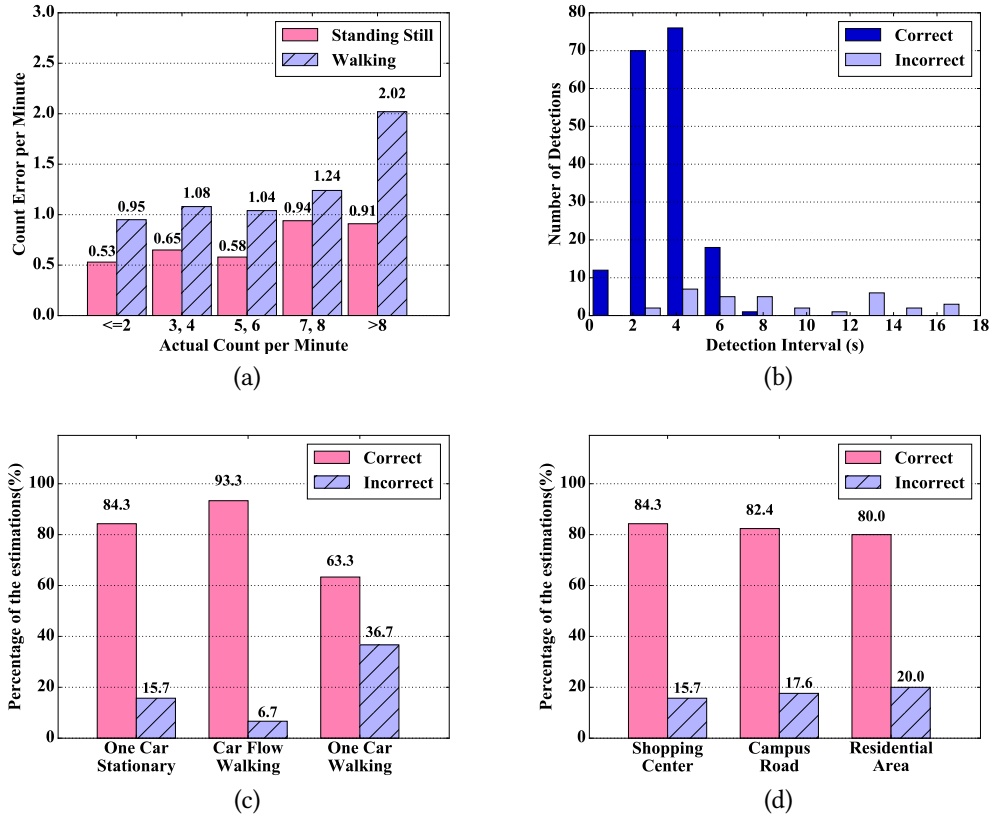


Fig. 9. (a) The average counting error is 0.53 for standing users and 0.95 car per minute for walking users, when the actual traffic ≤ 2 cars per minute. The average counting error increases to 0.58 for standing users and 1.04 for walking users when the ground truth is 5 or 6 cars per minute. (b) We correctly estimated the car driving direction with probability of 84% (177 out of 210) for all single car test traces. (c) We correctly estimated the driving direction of car flow and single with probability of 93% (28 out of 30) and 67%(20 out of 30), when the user is walking. (d) We correctly estimated the driving direction of cars with probability of 84.3%.

6.2 Driving Direction Estimation

Next, we evaluated the accuracy of our car direction estimation. In this use case, we used the same audio tracks that we collected in Sec. 6.1 in the first three settings. In our experimental setup, the user was walking on the sidewalk along the road, so the car was approaching either from the back or the front. For each audio track, our estimation can be correct or incorrect. We report the number of correct estimations or the correct estimation rate (the ratio between the number of correct estimations over the total number of audio tracks) to evaluate the estimation accuracy.

Setting I: Stationary User & Single Car: We first demonstrate that Auto++ can estimate a car's driving direction when the user is stationary. Fig. 9 (a) presents the histograms of correct and incorrect direction estimations (out of 210 total audio tracks) at different times. The results show that, Auto++ can correctly estimate the driving direction with 84.3% of the time when the car is 3.5 seconds away.

Setting II: Mobile User & Multiple Cars: Next, we evaluate Auto++'s direction estimation accuracy when multiple cars approach a walking user in Fig. 9 (b). When a user is walking, the SPL measures become less stable, and the correct estimation rate goes down, i.e., 63.3% in our case. However, when multiple cars follow each other to approach the walking user, Auto++ can estimate their direction with a much higher correct estimation rate, 93.3%.

Setting III: Noisy Environments: Finally, we show that Auto++ can even estimation the driving direction with a reasonable accuracy in various noisy environments. Fig. 9(c) shows that its estimation accuracy is similar across three scenarios, as well as on a quiet parking lot, suggesting our direction detection algorithm is not sensitive to environmental noise.

7 RELATED WORK

Car-Centered Context Sensing: In the area of pedestrians safety study, many works have been proposed to detect the presence of pedestrian proactively. Different sensing technologies such as piezoelectric sensor, ultrasonic detector, microwave radar can be deployed on the infrastructure or the cars [4]. Piezoelectric sensor requires the contact of the cars or the users, which limits its effective area. Ultrasonic detector and microwave radar are commonly used in the latest model of cars, but it remains a challenge to adopt these sensors in the old ones. Work by Tsuzuki et al. [32] proposed a machine learning based sound approaching detection technique for vehicles to detect other vehicles coming from their back. However, this approach has not been evaluated in noisy environment, and has not been validated on the pedestrian side where ambient noise and user movement pattern are different.

Another category of study focuses on using computer vision to detect the pedestrian. Work by Viola et al. [33] introduced a pedestrian detection platform, which could differentiate a pedestrian from the environment based on the walking motion. Dollar et al. [11] proposed a set of evaluation metrics to investigate pedestrian detection in most of the challenging scenarios. Although pedestrian detection based on computer vision are promising, their performance in the low light environment remains questionable.

Pedetrian-Centered Context Sensing: Besides car-centered methods, user-centered sensing techniques try to extract roadside context from the other angle. Work by Riaz et.al [26] proposed a hybrid system that utilizing wireless sensor network and GPS to reduce the vehicle/pedestrian collisions. This approach requires a scaled deployment, which is difficult to be generalized in realistic. Jain et.al [18] proposed a mobile system that detects the event when a user steps in an intersection. However, it requires the user to mount an additional sensor on the shoes, which adds extra cost and is unlikely to be ubiquitous. Takagi et al. [31] introduced a hybrid and electric vehicles detection system that focusing on switching sound generated by electric motor, but it fails short in detecting cars other than these two types of cars.

In contrast, Auto++ only relies on smartphone without requirement any infrastructure support. Also, it is more flexible and efficient, since microphone is capable of collecting audio data from any angle and audio processing costs less computing resource. Finally, it relies on the sound that generated by tire-road friction, which is common across different car types.

Sound Source Localization: The topic for signal localization is well addressed in the field of signal processing. Earlier work in this domain focused on using the signal difference among multiple receivers and the geometric structure of multiple receivers to locate the incoming signal. Schau et.al [28] propose a solution for source location bu giving time-of-arrival difference measurements when the distance from the source to any arbitrary reference is unknown. This technique was widely applied to military. Damarla et al. [10] developed a special receiver array device to localize the location of a sniper. Typical way to find the time-of-arrival (ToA) difference is to use cross correlation function in time domain[23]. However, due to the hardware limitation and insufficient temporal information in our system, we can not accurately locate the car using these techniques. Another work [39]

propose a method to find the ToA difference by using internal phase difference (IPD). This approach requires a high signal to noise rate, but the signal is heavily prone to the surrounding noise in our usage scenarios.

To detect a moving object with acoustic signal, Doppler shift is widely used in many studies. Work by Cheng et.al [8] proposed a system a system to localize an underwater moving object using acoustic sensor network. Chan et.al [7] proposed a system to localize a moving object by sending and receiving RF signals to calculate the Doppler shift. These method, however, require a pre-known distinctive frequency component of the signal, which is difficult to find in our scenario.

8 CONCLUSIONS

We have presented Auto++, an unsupervised approach for detecting approaching cars. Auto++ depends on non-obvious observation of the sounds that car tires make, and thus is applicable even to electric vehicles, which are otherwise quiet compared to gas powered cars. The novel feature Top-Right Frequency (TRF) extracted by blurred edge detector (BED), was able to detect a car with 91% accuracy even four seconds away. Further, our direction detection algorithm can detect the driving direction of approaching cars in most of the instances. Finally, Auto++ can provide car counting service with only 0.9 counting error per minute when the actual count per minute is beyond 8. We believe that the design, implementation and evaluation of our results present important practical contributions towards enabling and enriching more pedestrian-centric applications.

REFERENCES

[1] 2017. Apple Map. http://www.apple.com/ios/maps/. (2017). Accessed: 2017-01-31.

[2] 2017. Google Map. https://maps.google.com. (2017). Accessed: 2017-01-31.

[3] Massimo Bertozzi, Luca Bombini, Pietro Cerri, Paolo Medici, Pier Claudio Antonello, and Maurizio Miglietta. 2008. Obstacle detection and classification fusing radar and vision. In *Intelligent Vehicles Symposium, 2008 IEEE*. IEEE, 608–613.

[4] Fanping Bu and Ching-Yao Chan. 2005. Pedestrian detection in transit bus application: sensing technologies and safety solutions. In *Intelligent Vehicles Symposium, 2005. Proceedings. IEEE*. IEEE.

[5] John Canny. 1986. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), 679–698.

[6] Yiu Tong Chan and KC Ho. 1994. A simple and efficient estimator for hyperbolic location. *Signal Processing, IEEE Transactions on* 42, 8 (1994), 1905–1915.

[7] Yiu Tong Chan and JJ Towers. 1992. Passive localization from Doppler-shifted frequency measurements. *Signal Processing, IEEE Transactions on* 40, 10 (1992), 2594–2598.

[8] Wei Cheng, Andrew Thaeler, Xiuzhen Cheng, Fang Liu, Xicheng Lu, and Zexin Lu. 2009. Time-synchronization free localization in large scale underwater acoustic sensor networks. In *Distributed Computing Systems Workshops, 2009. ICDCS Workshops' 09. 29th IEEE International Conference on*. IEEE, 80–87.

[9] John Chon and Hojung Cha. 2011. Lifemap: A smartphone-based context provider for location-based services. *IEEE Pervasive Computing* 10, 2 (2011), 58–67.

[10] Thyagaraju Damarla, Lance M Kaplan, and Gene T Whipps. 2010. Sniper localization using acoustic asynchronous sensors. *Sensors Journal, IEEE* 10, 9 (2010), 1469–1478.

[11] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. 2009. Pedestrian detection: A benchmark. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE.

[12] Jon Froehlich, Mike Y Chen, Sunny Consolvo, Beverly Harrison, and James A Landay. 2007. MyExperience: a system for in situ tracing and capturing of user feedback on mobile phones. In *Proceedings of the 5th international conference on Mobile systems, applications and services*. ACM, 57–70.

[13] Fredrik Gustafsson and Fredrik Gunnarsson. 2003. Positioning using time-difference of arrival measurements. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*. IEEE.

[14] Jean-Françcois Hamet, Francis Besnard, Sonia Doisy, Joël Lelong, and Emmanuel Le Duc. 2010. New vehicle noise emission for French traffic noise prediction. *Applied acoustics* 71, 9 (2010), 861–869.

[15] John N Holmes, Wendy J Holmes, and Philip N Garner. 1997. Using formant frequencies in speech recognition.. In *Eurospeech*, Vol. 97. 2083–2087.

[16] Wenchao Huang, Yan Xiong, Xiang-Yang Li, Hao Lin, Xufei Mao, Panlong Yang, and Yunhao Liu. 2014. Shake and walk: Acoustic direction finding and fine-grained indoor localization using smartphones. In *INFOCOM, 2014 Proceedings IEEE*. IEEE.

[17] Keijiro Iwao and Ichiro Yamazaki. 1996. A study on the mechanism of tire/road noise. *JSAE review* 17, 2 (1996), 139–144.
[18] Shubham Jain, Carlo Borgiattino, Yanzhi Ren, Marco Gruteser, Yingying Chen, and Carla Fabiana Chiasserini. 2015. Lookup: Enabling pedestrian safety services via shoe sensing. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. ACM.
[19] Regina Kaune. 2012. Accuracy studies for TDOA and TOA localization. In *Information Fusion (FUSION), 2012 15th International Conference on*. IEEE, 408–415.
[20] Branislav Kusy, Akos Ledeczi, and Xenofon Koutsoukos. 2007. Tracking mobile nodes using RF doppler shifts. In *Proceedings of the 5th international conference on Embedded networked sensor systems*. ACM.
[21] Nicholas D Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T Campbell. 2010. A survey of mobile phone sensing. *IEEE Communications magazine* 48, 9 (2010).
[22] Sugang Li, Ashwin Ashok, Yanyong Zhang, Chenren Xu, Janne Lindqvist, and Macro Gruteser. 2016. Whose move is it anyway? Authenticating smart wearable devices using unique head movement patterns. In *Pervasive Computing and Communications (PerCom), 2016 IEEE International Conference on*. IEEE, 1–9.
[23] Jian Liu, Yan Wang, Gorkem Kar, Yingying Chen, Jie Yang, and Marco Gruteser. 2015. Snooping keystrokes with mm-level audio ranging on a single phone. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM.
[24] Suman Nath. 2012. ACE: exploiting correlation for energy-efficient and continuous context sensing. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 29–42.
[25] Lawrence R Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* 77, 2 (1989), 257–286.
[26] Z Riaz, DJ Edwards, and A Thorpe. 2006. SightSafety: A hybrid information and communication technology system for reducing vehicle/pedestrian collisions. *Automation in construction* 15, 6 (2006), 719–728.
[27] Eduardo Romera, Luis M Bergasa, and Roberto Arroyo. 2015. A real-time multi-scale vehicle detection and tracking approach for smartphones. In *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*. IEEE, 1298–1303.
[28] HC Schau and AZ Robinson. 1987. Passive source localization employing intersecting spherical surfaces from time-of-arrival differences. *Acoustics, Speech and Signal Processing, IEEE Transactions on* 35, 8 (1987), 1223–1225.
[29] Anmol Sheth, Srinivasan Seshan, and David Wetherall. 2009. Geo-fencing: Confining Wi-Fi coverage to physical boundaries. In *International Conference on Pervasive Computing*. Springer, 274–290.
[30] Frédéric Suard, Alain Rakotomamonjy, Abdelaziz Bensrhair, and Alberto Broggi. 2006. Pedestrian detection using infrared images and histograms of oriented gradients. In *Intelligent Vehicles Symposium, 2006 IEEE*. IEEE, 206–212.
[31] Masaru Takagi, Kosuke Fujimoto, Yoshihiro Kawahara, and Tohru Asami. 2014. Detecting hybrid and electric vehicles using a smartphone. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 267–275.
[32] Hirofumi Tsuzuki, Mauricio Kugler, Susumu Kuroyanagi, and Akira Iwata. 2010. A novel approach for sound approaching detection. *Neural Information Processing. Models and Applications* (2010), 407–414.
[33] Paul Viola, Michael J Jones, and Daniel Snow. 2005. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision* 63, 2 (2005), 153–161.
[34] Tianyu Wang, Giuseppe Cardone, Antonio Corradi, Lorenzo Torresani, and Andrew T Campbell. 2012. WalkSafe: a pedestrian safety app for mobile phone users who walk and talk while crossing roads. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*. ACM.
[35] Christian Wojek, Stefan Walk, and Bernt Schiele. 2009. Multi-cue onboard pedestrian detection. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 794–801.
[36] Xinzhou Wu, Sundar Subramanian, Ratul Guha, Robert G White, Junyi Li, Kevin W Lu, Anthony Bucceri, and Tao Zhang. 2013. Vehicular communications using DSRC: challenges, enhancements, and evolution. *IEEE Journal on Selected Areas in Communications* 31, 9 (2013), 399–408.
[37] Chenren Xu, Sugang Li, Gang Liu, Yanyong Zhang, Emiliano Miluzzo, Yih-Farn Chen, Jun Li, and Bernhard Firner. 2013. Crowd++: Unsupervised speaker count with smartphones. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*. ACM, 43–52.
[38] Qing Xu, Tony Mak, Jeff Ko, and Raja Sengupta. 2004. Vehicle-to-vehicle safety messaging in DSRC. In *Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*. ACM, 19–28.
[39] Wenyi Zhang and Bhaskar D Rao. 2010. A two microphone-based approach for source localization of multiple speech sources. *Audio, Speech, and Language Processing, IEEE Transactions on* 18, 8 (2010), 1913–1928.
[40] Fang Zheng, Guoliang Zhang, and Zhanjiang Song. 2001. Comparison of different implementations of MFCC. *Journal of Computer Science and Technology* 16, 6 (2001), 582–589.

Received February 2017; revised May 2017; accepted September 2017