BaroSense: Using Barometer for Road Traffic Congestion Detection and Path Estimation with Crowdsourcing

ANUJ DIMRI and HARSIMRAN SINGH, University of Utah, USA
NAVEEN AGGARWAL, UIET, Panjab University, India
BHASKARAN RAMAN, IIT Bombay, India
K. K. RAMAKRISHNAN, University of California, Riverside, USA
DIVYA BANSAL, PEC University of Technology, India

Traffic congestion on urban roadways is a serious problem requiring novel ways to detect and mitigate it. Determining the routes that lead to the traffic congestion segment is also vital in devising mitigation strategies. Further, crowdsourcing this information allows for use of these strategies quickly and in places where infrastructure is not available. In this work, we present an unconventional method, using the barometer sensor of mobile phones to (a) detect road traffic congestion and (b) estimate the paths that lead to the congested road segment. We make the observation that roads are not completely flat and very often, altitude varies along the road. The barometer sensor chips are sensitive enough to measure these variations and consume very little energy of the mobile phone, compared to other sensors such as the GPS or accelerometer. We devise a feature set to map the rate of change of this altitude as the user moves into activities characterized as "still" and "motion," which are further used by the traffic congestion detection algorithm (RoadSphygmo) to classify the group of users as being in "moving," "congestion," or "stuck" states. To estimate the paths that lead to the congested road segment, we compare the user's barometer sensor readings with a pre-stored road signature of barometer values using Dynamic Time Warping (DTW). We show that by using correlation of barometer sensor values, we can determine if users are in the same vehicle. We crowdsource this information from multiple mobile phones and use majority voting technique to improve the accuracy of traffic congestion detection and path estimation. We find a significant increase in the accuracies using crowdsourced information as compared to individual mobile phones. Further, we show that we can use barometer sensor for other applications such as bus occupancy, boarding/deboarding of a vehicle, and so on. The validation of the state determined by RoadSphygmo is done by comparing it with average GPS speed calculated during the same time period. The path estimation is validated over different intersections and considering various cases of commuter travel. The results obtained are promising and show that the traffic state determination and the estimation of the path taken by the commuter can achieve high accuracy.

CCS Concepts: • **Information systems** → **Mobile information processing systems**;

Anuj Dimri and Harsimran Singh contributed equally to the paper.

This work was done as part of the project "CARTS—Communication Assisted Road Transportation System," funded by ITRA, Media Lab Asia and Design Innovation Center, Panjab University, Chandigarh, India.

Authors' addresses: A. Dimri and H. Singh, University of Utah, 201 Presidents Circle, Salt Lake City, UT, 84112; emails: {anuj, hsingh4}@cs.utah.edu; N. Aggarwal, UIET, Panjab University, Chandigarh Rd, Sector 25, Chandigarh, Chandigarh, India; email: navagg@gmail.com; B. Raman, IIT Bombay, Main Gate Rd, IIT Area, Powai Mumbai, Maharashtra, India; email: br@cse.iitb.ac.in; K. K. Ramakrishnan, University of California, Riverside, 900 University Ave, Riverside, California; email: kk@cs.ucr.edu; D. Bansal, PEC university of Technology, Vidya Path, Sector 12, Chandigarh, Chandigarh, India; email: divya@pec.edu.in.

1550-4859/2019/11-ART4 $15.00
https://doi.org/10.1145/3364697

Additional Key Words and Phrases: Activity recognition, barometer sensor, crowdsourcing, path estimation, smartphones, traffic congestion detection

ACM Reference format:
Anuj Dimri, Harsimran Singh, Naveen Aggarwal, Bhaskaran Raman, K. K. Ramakrishnan, and Divya Bansal. 2019. BaroSense: Using Barometer for Road Traffic Congestion Detection and Path Estimation with Crowdsourcing. *ACM Trans. Sen. Netw.* 16, 1, Article 4 (November 2019), 24 pages.
https://doi.org/10.1145/3364697

1 INTRODUCTION

Road traffic congestion is a major problem that continues to worsen everywhere, but especially in developing countries. Its consequences include adverse affects on environmental and significant impact on people's health, besides the economic consequences including but not limited to wasted fuel and individual productivity loss. Multiple road segments may be in the vicinity of the congested location, but not all of them may be congested. Thus, for an effective mitigation strategy it is important to detect the exact road segment a user traveled to reach the congested segment/point. If this can be done in real time, then the information can potentially be disseminated to other users to enable adoption of congestion mitigation strategies (an example is shown in Figure 1). Further crowdsourcing from multiple users can help increase the accuracy of detecting traffic congestion and path estimation. Also keeping the privacy of the user in mind, the user should not be tracked throughout his/her journey. Thus, crowdsourcing the road congestion information only when the user encounters stuck state can be a low-cost privacy-preserving approach for such detection and mitigation strategies.

Researchers have proposed both infrastructure-based and infrastructure-less solutions to detect the current state of traffic on roads. Infrastructure-based techniques such as using a camera [17], inductive loop [23], magnetic sensor [6], RF-based solutions [21], and so on have been applied but have significant initial setup cost, regular maintenance, and are constrained by the coverage limitations of the sensor used. Infrastructure-less solutions mainly involve the use of smart phone sensors such as GPS [24], GSM [13], and accelerometers [15]. It is relatively easy to glean information from these sensors. However, GPS draws a large amount of power (170mW–630mW) depending upon usage and has unreliable performance [19] in the absence of stable signals. To encourage users to participate in the crowdsourcing for societal good, it is important and necessary that information be collected in an energy-efficient way with little user intervention. The GPS sensor, in combination with Wi-Fi sensors and other historical data for traffic congestion, has been used in some of the more straightforward approaches. However, it can be less effective, as users tend to not turn these sensors on, because they quickly drain the battery. Moreover, in developing countries, where a significant portion of the traffic is two-wheelers, three-wheeled auto-rickshaws, and public buses, a user's ability to charge their cell phone periodically is extremely limited. We believe a significant opportunity is gained by collecting data from commuters who are using public transport, such as buses for their day-to-day travel. These commuters can only be engaged if sensors collecting the data are not power-hungry. Thus, the availability of low-cost sensors on users' mobiles will be a big plus. In this article, we propose BaroSense, a system for road congestion detection and path estimation using the barometer sensor. Unlike GPS, the barometer sensor uses very little battery power of the phone.

Barometer sensors, a recent addition to smart phones, are increasingly used for indoor localization, such as tracking the floor level [14], weather analysis [4], and estimating a user's location on a subway [12]. The recent use of barometer for context detection is explored by Sankaran et al.

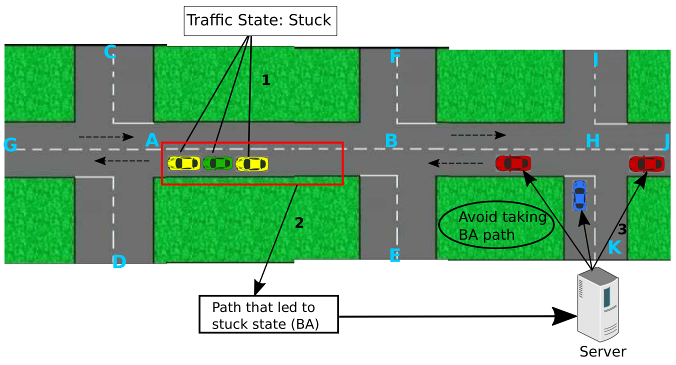

Fig. 1. BaroSense aim: Detecting congestion and estimating the path taken by vehicle prior to congestion.

[20]. They classify the user movement using variation in altitude values measured using barometer sensor. We enhance their proposed feature of "jumps" by choosing an appropriate window size and threshold. We found a single window does not reliably classify user activity. We then build a feature set by choosing a batch of jumps, the standard deviation of altitude in the window, and further use different classifiers for user activity recognition. This is then used by our road traffic congestion detection algorithm (RoadSphygmo). Further, we even estimate the most recent path traveled by the commuter to reach the congested location. Path estimation is done only when our traffic congestion algorithm predicts congested traffic states.

Roads may seem perfectly flat to the naked eye, but in reality the altitude varies at different points all along the length of the road. For example, we see a substantial change in the height of the road by examining the altitude using Google's Elevation API. The barometer sensor is sensitive enough to measure these altitude changes. In a given period of time, a vehicle on a free-flowing road will cover a larger distance as compared to a vehicle on a congested road, and thus see a corresponding larger altitude variation. This forms the basis of our traffic congestion algorithm. Further, these altitude changes at different points of a road are different for each road. Fingerprinting these changes (termed as road signature) allows us to identify a road given a set of barometer sensor readings. This forms the basis for our path estimation approach.

The barometer sensor, which is a low-power non-private passive sensor, consumes very little power as compared to other sensors like accelerometer on a user's cell phone. In a Samsung Galaxy S3, a barometer sensed at 1Hz sees only 2% increase (to 110mW) in base power (which is 108mW). Comparatively, the accelerometer that is normally used at 20Hz even at 2Hz results in a 51% increase (164mW) in base power [20]. When compared with GPS, the power consumed by the barometer sensor is typically less than 10μW [22], whereas the power consumed by the GPS module GlobalSat EM506 is 170mW [8]. If we sought to limit the additional power consumption to 10μW by duty-cycling the GPS, this would only allow for one reading every four hours [11]. Using GPS with a small duty cycle—say, every five seconds—is also not a good idea, as it takes some time to get the GPS lock before getting useful information. Using GPS with a large duty cycle does not serve our purpose, as the commuter would cover multiple road segments and we would not be able to estimate the path that led to the congested traffic location. So, that means we would have to keep the GPS on continuously to get useful information. This would come at the cost of the mobile device's battery being drained quickly. More recent versions of accelerometers consume 4μA of power [2]. However, setting the configuration of the accelerometer to achieve maximum accuracy/performance increases the power consumption to 14.5μA [1]. In contrast, newer barometers consume 1.7μA of power at 1Hz [3]. We believe this is very suitable for our purpose. The barometer sensor is also orientation-independent. Even when subjected to oscillations of up to 1cm, it continues to give the correct pressure reading [20].

Ho et al. [11] used the barometer sensor to predict path of a complete journey of the user and they do so in 7–15 minutes on an Intel i7 machine. Such large time scales would defeat our purpose of detecting congested path in real time so oncoming users can avoid taking this congested path. So, we propose a method to build a database that facilitates a quick lookup of a limited number of potential road segments in the proximity of the user rather than naively comparing all the paths in the database. This limited number of potential paths are then compared with the user's barometer sensor readings. The detailed methodology is discussed in Section 3 and Section 5.

Based on an understanding of the patterns in the raw barometer sensor readings, we classify the movement of a vehicle as *"still"* or in *"motion."* A set of vehicle movements observed over a period of time is used to decide the traffic state the user is in by the traffic congestion detection algorithm. The traffic congestion detection algorithm classifies the traffic state into *"moving," "congestion,"* or *"stuck."* If the traffic state is *"stuck,"* then the path traveled by the vehicle is estimated and this information can be disseminated to other users. Each road has different variation in altitude and pressure values, thus making each road unique. This pattern of variation in altitude values along the length of the road is the road's signature. The path estimation of the vehicle is done by comparing the road signatures stored in the database with the barometer sensor readings from a user's (or a set of users') mobile device. Varying speed of the vehicle, different lengths of the roads make the sequences for comparison to be of unequal lengths. Dynamic Time Warping (DTW) is capable of comparing these unequal length sequences effectively. Thus, we use it to determine the similarity of the two sequences (road signature in the database and the user's barometer readings) [27]. DTW is a time series alignment algorithm that uses dynamic programming to find the minimum DTW cost path between two time-dependent sequences. Of all the possible candidate paths the vehicle could have taken, the most similar road signature to the user's barometer readings is considered for estimating the path of the user's vehicle. We observed that users traveling the same stretch of the road will observe similar changes in the barometer pressure readings and have high correlation values. This can be exploited to do majority voting of the crowdsourced data to obtain increased accuracy of the traffic congestion detection algorithm and path estimation. It can also be used for detecting activities such as boarding and deboarding of a vehicle (bus), occupancy rate of the bus, and so on.

Since the pressure at a certain location is affected by environmental changes, we need to carefully consider the effect it may have on pressure readings. Over the period of a day, the ambient temperature changes. So does the pressure. But the change is gradual and spans a period of several hours. Our method considers values in a time period ranging from a few seconds to a couple of minutes—a very small duration for these environmental changes to affect our approach. There are a number of other practical considerations with regard to a user environment that we need to take into account. The phone may be in a handbag or pocket of the user, the air conditioning of the vehicle (which affects the pressure) may be turned on or off, the windows may be open or closed, and so on. Being cognizant of these scenarios, we performed multiple experiments considering combination of all the above conditions. We found that barometer sensor is robust in these conditions. Also, events like the AC being turned on/off have a transition time of a few seconds, during which they contribute noise to the data only for a couple of seconds. Our sliding window approach along with the use of relative values quickly helps to adapt to the new environment.

The main contributions of the article are summarized as follows:

- Based on an understanding of the patterns in raw barometer readings, we propose a feature set to classify user movement activity into two broad classes: *"still"* and *"motion"* (activity recognition) in diverse environmental conditions.

- Based on this activity recognition, a congestion detection algorithm classifies the user's current traffic state into *"moving," "congestion,"* and *"stuck."* The average accuracy of our algorithm to determine these three traffic states is 80.37%, 70.84%, and 97.36%, respectively.
- A methodology to build a database of road signatures that permits a quick lookup of potential road segments a user could be on. An approach to estimate the recent path traveled by the user before encountering the "stuck" state by comparing the real time barometer readings of the user with pre-stored road signatures in the database.
- Demonstration of the utility of correlation of barometer values to identify the users traveling along the same stretch of the road. We also show how it can be used for other applications such as detecting the boarding and deboarding of a vehicle, occupancy rate of a vehicle, and so on.
- A crowdsourcing system that does majority voting of the users in the same area to improve the accuracy of traffic congestion detection and path estimation.
- Validation of our approaches by performing multiple experiments on both personal and public vehicles—such as city buses and three-wheeled auto-rickshaws across different cities using multiple phones.

This article presents an enhanced version of RoadSphygmo [7]. We improved the classifier by using a new enhanced feature set and trained it on data from a diverse set of vehicles, including public buses, personal cars, taxis, and three-wheeled auto-rickshaws. This led to an increase in the accuracy of the "moving" and "congested" states (when compared with Reference [7]). Additionally, the path estimation module (composing the methodology to build a database of road signatures that enables the quick lookup and DTW-based road signature matching algorithm) and correlation-related applications, such as detecting the boarding and deboarding of a vehicle and estimating the occupancy rate of a vehicle, are entirely new contributions of this article.

The rest of the article is organized as follows: Section 2 discusses related work, Section 3 contains the BaroSense system overview, which describes the components. Section 4 presents RoadSphygmo (our traffic congestion detection algorithm). Section 5 describes the path estimation strategy followed by Section 6, which describes the approach of crowdsourcing in detail. Section 7 presents the conclusion and future work.

2 RELATED WORK

Related work is organized into two parts: The first part describes different activities for which barometer sensor is being used. The second part describes different state-of-the-art techniques to detect traffic congestion and its location.

Use of barometer sensor for other applications: Barometer is used for weather prediction [4], aiding GPS [30], indoor localization [18], and tracking floor levels [14]. It is also used to distinguish between stairs and elevators using vertical speeds. Sankaran et al. [20] use barometer for context detection of user activities such as idling, walking, and vehicle. They do so at very low power and consume 3 mW power over base power, as opposed to 35 mW power over base power used by the accelerometer-based approach. In this work, we expand their idea of "jumps" along with changes in the window sizes suitable for our purpose of traffic congestion detection. Hyuga et al. [12] use barometer to estimate a user's location on a subway. They determine whether the train is stopped or running and use this information to estimate the exact station where the train stopped using the actual elevation of the station. Ho et al. [11] uses barometer sensor to determine the complete path of the user. They make use of weather station data, Open Street Map (OSM), and Google Elevation API to build the database. Their dynamic programming–based algorithm takes 7–15 minutes for 92 sq km map on an Intel i7 processor to do so. Adopting their approach for path estimation defeats our purpose of devising a real-time method to disseminate information to interested users. Therefore, we are only interested in the most recent segment traveled by the user, as it is the segment that leads to the congested area. Moreover, in developing countries OSM and Google Elevation API do not have reliable resolution. Also in road traffic congestion, only the most recent segment of the user's path is of concern, as it is the segment that led to traffic congested point.

Detecting traffic congestion and its location: To detect traffic congestion on roads, different infrastructure-based solutions are proposed where sensors are installed on the roadside. Some of the common infrastructure-based solutions are: vision-based cameras [17], inductive loops [23], magnetic sensors [6], and so on, on the roads. Tyagi et al. [26] use acoustic signals from microphones installed on roadside to detect traffic density. Infrastructure-based solutions are associated with initial setup cost, regular maintenance, and are affected by the coverage limitation of the sensor.

Sensors present in mobile phones have also been used extensively to detect traffic congestion. Herrera et al. [10] use GPS enabled mobile devices to determine traffic congestion from real-time velocity of traffic flow. Yoon et al. [29] identify traffic conditions on a city's streets given location traces collected from on-road GPS devices. But GPS requires large amount of power for functioning and loses signals frequently, thus making it unsuitable to be used on mobile phones. As battery issues are critical, users tend to not switch on the GPS to save battery, thus limiting its use. Cellular signal–based techniques are also used to detect road traffic congestion. Thajchayapong et al. [25] used the concept of cell dwell time to estimate traffic congestion, while Janecek et al. [13] use cellular data collected from cellular mobile networks to infer about the state of the traffic. Cellular signals require sufficient cellular towers for their functioning; however, barometer sensor is independent of any external infrastructure. Hence, it can be used as a more reliable option as compared to cellular signal–based techniques.

Lv et al. [15] propose a system that uses accelerometer and cellular signal to detect the degree of congestion on the traveled road segment. Accelerometer is used for movement detection, cellular signal is used to determine the road segment traveled. They further infer the degree of congestion (fluency, light congestion, and heavy congestion) on the traveled road segment. We use only barometer sensor to classify traffic state into *"moving," "congestion,"* and *"stuck."* To tell the user's path, Han et al. [9] use accelerometer data to select a subset of possible candidate paths from all the paths in a given map. We instead use barometer data of the roads to estimate the path of the user from all the possible candidate paths.

Won et al. [28] estimate the significant journeys of a user using barometer sensor only. They even predict door-open and -close events of a vehicle based on barometer readings. However, they estimate the complete paths, and we are only concerned with those paths that led the commuter to congested areas. Estimating the complete path of the user can raise serious privacy issues. We tie traffic congestion detection with our path estimation technique and further use crowdsourcing to increase our accuracy, which to the best of our knowledge has never been studied before in this context.

3 BAROSENSE: SYSTEM OVERVIEW

The BaroSense system is composed of two main components as shown in Figure 2:

(1) **User mobile device side/Client side:** The traffic congestion detection component runs on the user's mobile phone and broadly classifies the state of the road traffic as *"moving," "congestion"* or *"stuck"* based on the readings obtained from the barometer sensor. On encountering a stuck/congested state, the user's device sends its current location and the barometer readings to the server for path estimation.

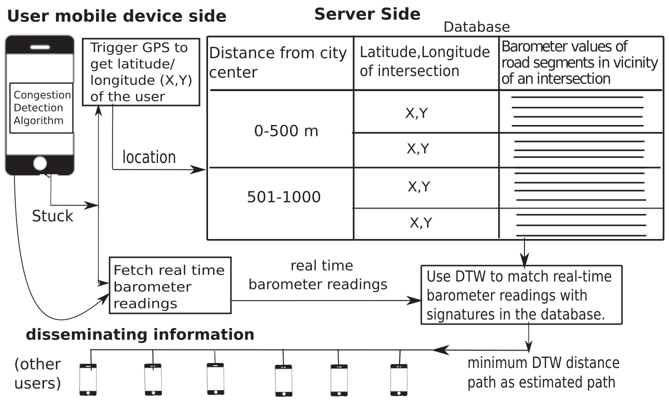
User mobile device side
Server Side
Database
Congestion Detection Algorithm
Trigger GPS to get latitude/ longitude (X,Y) of the user
location
Stuck
Distance from city center
Latitude,Longitude of intersection
Barometer values of road segments in vicinity of an intersection
0-500 m
501-1000
X,Y
Fetch real time barometer readings
real time barometer readings
Use DTW to match real-time barometer readings with signatures in the database.
disseminating information
(other users)
minimum DTW distance path as estimated path

Fig. 2. BaroSense: System Overview.

(2) **Server side:** A centralized server component builds a dynamic congestion map of the city from the crowdsourced data from multiple users by estimating the path traveled by the user before coming to *"stuck"* state (as it is the path that led the user to the congestion point).

3.1 User Mobile Device Side/Client Side

Using a pattern of raw barometer readings collected on a user's mobile phone, we classify the activity of the user into *"still"* and *"motion"* class using Support Vector Machine (SVM). A set of these user activities is further used by our traffic congestion algorithm to determine the current traffic state (*"moving," "congestion,"* and *"stuck"*) of the user. On having encountered a stuck state it becomes important to determine the path the user took that led to the congested area. An entire city can have a large number of road segments. Comparing with all the road segments can take a large amount of time and defeat our purpose of real-time congestion detection and subsequent dynamic congestion map generation. Therefore, to quickly narrow down on the potential road segments, we turn on GPS momentarily to get a coarse location of the user and then turn the GPS off. Without this coarse location, the naive algorithm will iterate over all the path signatures in the database and try to find a match. It is important to note that the GPS is turned on briefly, e.g., if the user's journey is of 3 hr, we do not keep the GPS on for the entire journey. Only when the congestion detection algorithm encounters stuck state, the GPS is used to get the coarse location. User's location along with real-time barometer readings are passed to the server for further computation.

3.2 Server Side

On obtaining the coarse location of the user on the server side, we first need to narrow down the potential road segments that the user could have traveled to reach the congested location. To achieve this, there are two possible indexing mechanisms: using indices in an X-Y grid, or using polar coordinates (distance from city center). It is an implementation choice and we have chosen the latter option because of the ease of implementation, as the database key is a single number, i.e., distance from the city center. Given the user's coarse location, we calculate the distance of the user's location from the city center. This distance narrows our search for the possible intersections by telling only that set of intersections that lie in the same distance range from the city center as the user does. Each intersection is represented by its location (X,Y), where X and Y correspond to latitude and longitude of the intersection, respectively. Once we get to know this set of intersections, we find the distance of the user from all the intersections in that set and the one with the minimum distance is chosen as the nearest intersection. Then all the road signatures corresponding to this nearest intersection will be compared with the barometer sensor values that are obtained from the user. Due to varying speed of the user and different length of the roads, the real-time barometer values and road signatures in database can be of different length. We propose to use Dynamic Time Warping (DTW) [27] to compare these sequences. DTW is a time series alignment algorithm that uses dynamic programming approach to find the minimum cost path between the two sequences. The road segment for which the DTW-normalized distance is minimum is the estimated path that the user traveled to reach the congested location.

This information can be disseminated to interested users in real time, which can help in devising mitigation strategies such as choosing alternate routes, redirecting traffic, and so on. We perform all the above computations using relative values of barometer sensor readings to make the road signatures independent of the absolute pressure value of a particular day. Crowdsourcing the barometer data from multiple users helps increase the overall accuracy of the system, e.g., using majority voting of users traveling on the same road stretch based on the correlation values, we can improve the accuracy of congestion detection algorithm. Moreover, with the increase in the number of users pointing to a particular road segment being congested, the system's confidence in that particular road segment being congested grows.

To set the threshold for our traffic congestion algorithm (RoadSphygmo), we set aside a part of total collected data and perform empirical analysis on a range of values before choosing the appropriate threshold. The threshold details are covered in the next section. However, we found out during our evaluation on real-time data readings and elevation data from Google Elevation that the same threshold values work well in other places that have different terrains. This helps us to conclude that our algorithm works well even without recalibration in other cities.

4 TRAFFIC CONGESTION DETECTION

In BaroSense, traffic congestion detection operates by first classifying the activity as *"still"* or *"motion"* and subsequently observing a sequence of activity states to classify traffic state as *"moving," "congestion,"* or *"stuck."* We describe these in turn.

4.1 Activity Recognition

If the user is in a *"still"* state the pressure experienced by the barometer sensor does not change rapidly in a short duration of time. However, if the user is in a *"motion"* state the pressure would change continuously due to the elevation changes that occurs along the road stretch on which the user is traveling.

The pressure values are recorded every second and are converted to altitude using the formula proposed by National Oceanic and Atmospheric Administration [16]:

$$height = 44330 * \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right).$$

Height is in meters, p is the air pressure logged by the barometer sensor, and p_0 is a constant (sea level air pressure value, 1013.25 hpa).

To get the measure of altitude change of the user, the difference of altitude values 5 seconds apart is calculated. If the magnitude of the altitude change is more than a certain threshold it is called a "jump" [20]. This threshold is a configurable parameter depending on the terrain of the land and can be found by experimental means. We tried different values of threshold in the range of 0.5–1.5 meters and found the threshold in the range of 0.80–1 meter suitable for our purpose. We also observed that these threshold values were the same for the cities of Chandigarh and Mumbai (just

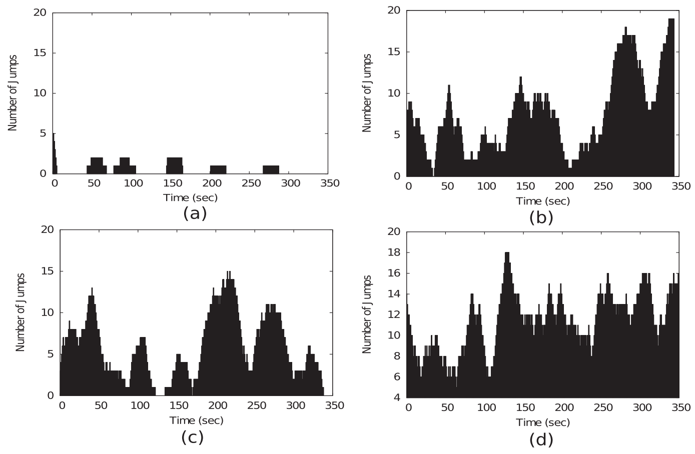

Fig. 3. Number of jumps under different experimental conditions. (a) Vehicle in still state (b) All windows open (c) All windows closed and AC off (d) All windows closed and AC on and phone in pocket [in (b), (c), (d), vehicle is in motion state].

as it was observed by Sankaran et al. [20]). In newer phones with better barometer sensor chips having improved resolution and noise filter, pressure measurements are more accurate. Hence, a lower threshold value can be used for new phones. During our experiments, the threshold in older barometer phones such as Google Nexus 5, Samsung S3, and Samsung Note 3 was set to 1 meter, whereas the threshold in new barometer phones such as Google Nexus 5x, Xiaomi Mi4, and HTC One M8 was set to 0.80 meter. It was observed that in case of *"motion,"* the vehicle tends to cover larger stretches of road and hence observes more jumps than when the vehicle is *"still."* Hence, to distinguish between *"still"* and *"motion"* activity, we calculated the number of jumps observed in a fixed sized time window. To detect the state of traffic the user is in reliably, and as quickly as possible, we want the window size to be small. Thus, the size of window is chosen as 20 seconds so sufficient number of jumps can be observed to be able to distinguish between *"still"* and *"motion."* A typical pattern of number of jumps in a 20-second window during *"still"* activity and different environments possible in a vehicle in *"motion"* activity are shown in Figure 3. Even though the pattern of jumps is not same in different environment conditions when the vehicle is in *"motion,"* in all the cases the number of jumps are significantly larger than *"still"* state. Thus, even under different environment conditions, we will be able to classify the user activity into *"still"* and *"motion."*

But the number of jumps in a single time window (denoted as w-jumps) can be misleading in detecting the activity of the user as *"still"* or *"motion."* For example, if the w-jumps is 3, it is hard to tell by looking at 3 whether it belongs to a vehicle when it is still or in motion. But, if we consider a pattern of w-jumps like 2,3,0,1...4, then we can say with high confidence that the activity is *"still,"* whereas if we consider a pattern 12,16,3,8,...12, then we know that the activity is *"motion"* in spite of the presence of 3 in the pattern above. Therefore, we need n such w-jumps to correctly classify the activity of the user. This pattern is obtained by sliding the 20-second window n times. If n is large, it may cover multiple activities; however, if n is too small, it may result in activity not being classified correctly. When we tested the classifier using different values of n, it was observed that $n = 10$ gave best results [7] to correctly classify the activity of the user.

Further, we observed that a user in *"motion"* state will have more standard deviation in the altitude values as compared to a user who is in *"still"* state. This observation led us to include the

Table 1. SVM Accuracy on Various Phones

Google Nexus5	Samsung S3	Samsung Note3	Google Nexus5x	Xiaomi Mi4
97.54%	93.62%	97.7%	98.72%	98.02%

standard deviation in the feature vector, which is calculated for the same time interval in which pattern of n w-jumps is considered.

We tested various classifiers to classify the user activity into *"still"* and *"motion"* class. Decision tree is the simpler one and computationally inexpensive. But it has a tendency of overfitting in the absence of large and reliable training set. Neural networks were not used, because they require extraction of large feature set, which is not computationally possible on mobile phones and would defeat the purpose of devising a low-power technique for traffic congestion detection.

Support Vector Machines (SVM) [5] are computationally less expensive for binary classification to classify user movements as *"still"* or *"motion."* Due to non-linear nature of feature vector, SVM with Radial Basis Function (RBF) kernel is used to reduce the computational requirement and map data as linearly separable. To further make the SVM more effective, we did tuning of the SVM model with C and gamma parameters, where C is the parameter for soft margin cost function and gamma is the RBF parameter. SVM was tuned by taking different combination of values of C and gamma parameters and the values that gave least error.

We collected 40 hours of *"motion"* data and 35 hours of *"still"* data using multiple phones in Chandigarh and Mumbai in India. To generate training and testing data, we randomly chose two-thirds of our data as training and the rest as testing. The data for the *"motion"* class must be collected when the vehicle is moving freely without any stops. So, the data was collected in free-flowing traffic and highways where the vehicle's speed is close to 60km/hr. This was done so we do not include any data when the vehicle was in *"congestion"* or *"stuck"* to be treated as *"motion"* in the training set. Similarly, in case of *"still"* the user will not experience any *"motion"* and hence remain at a particular spot. So, the training data for *"still"* class was collected with the phones being kept at a particular place.

We tested the SVM classifier with RBF kernel on different smart phones, and average accuracy obtained to detect the user activity in each case is shown in Table 1. It is quite evident from the results that newer phones with better chips gave slightly better accuracies. We also tested our training model on data obtained from Sankaran et al. [20]. Since their data only had readings for vehicles and subways, we only tested for *"motion"* class. We got an accuracy of 76.58%.

4.2 Traffic Congestion Detection

Once the user activity is classified as *"still"* or *"motion,"* a batch of **k** such activity states is used to determine the state of road traffic as *"moving," "congestion,"* or *"stuck."* With an aim to detect the state of traffic in roughly 2 minutes and give update on the new state of the traffic within 1 minute, we experimented with different values of k (1, 5, 10, 15, 20) and got best accuracy when **k** was 10 [7]. Each batch of 10 activity states is sent as an input to the traffic congestion detection algorithm named RoadSphygmo.

Algorithm 1 describes the RoadSphygmo algorithm in detail. It contains PredictTrafficCongestion procedure that receives 10 feature vectors as input and it calls the Activity_Classify procedure that outputs the activity as *"still"* or *"motion"* based on one feature vector. According to the output given by Activity_Classify procedure *still_count* and *moving_count* are incremented. This is repeated 10 times. If the *still_count* is greater than equal to 8, then traffic state is said to be *"stuck"*;

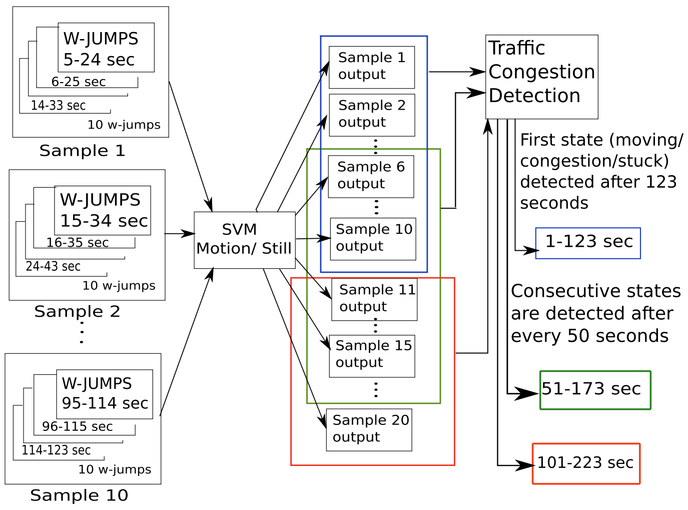

Fig. 4. First traffic state after 123 seconds and 50 seconds update for every successive state.

ALGORITHM 1: RoadSphygmo algorithm

```
1: procedure PredictTrafficCongestion(feature_vectors)
2:     still_count ← 0
3:     moving_count ← 0
4:     traffic_state ← Nil
5:     for i = 1 To 10 do
6:         prediction[i] ← Activity_Classify(feature_vectors[i])
7:         if prediction[i] == Still then
8:             still_count++
9:         else
10:            moving_count++
11:        end if
12:    end for
13:    if still_count >= 8 then
14:        traffic_state ← Stuck
15:    else if moving_count >= 5 then
16:        traffic_state ← Moving
17:    else
18:        traffic_state ← Congestion
19:    end if
20:    return traffic_state
21: end procedure
```

if the *moving_count* is greater than equal to 5, then traffic state is said to be *"moving."* If none of the above two conditions are satisfied, then the traffic state is said to be *"congestion"* state.

With the value of **k** set to 10, and considering the last 50% of the activities from the previous batch, we get the first state of traffic in 123 seconds and the state of the traffic gets updated every 50 seconds, as shown in Figure 4. Jumps are calculated in a 20-second window. A pattern of 10 such jumps makes a sample. The classifier classifies each sample as *"still"* or *"motion."* A batch of

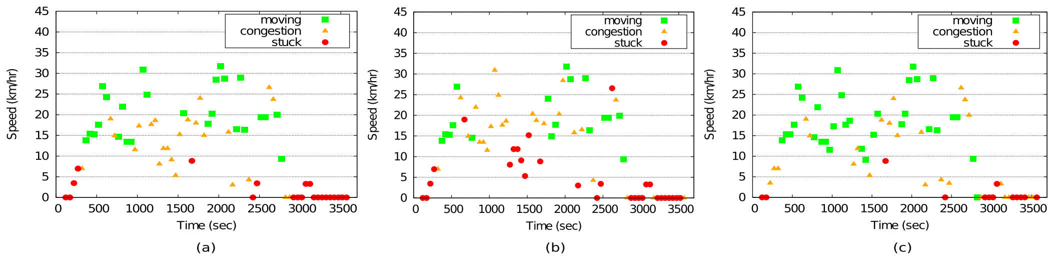

Fig. 5. Comparison between the speed from GPS and output from congestion algorithm (RoadSphygmo) during a bus journey using different mobile phones. (a) Nexus 5 (b) Nexus 5x (c) Xiaomi Mi4.

Table 2. Traffic State Accuracies for Nexus 5, Corresponding to Figure 5(a)

	Moving (20+ km/hr)	Congestion (10–20km/hr)	Stuck (0–10km/hr)
Moving	**57.70%**	38.46%	3.84%
Congestion	13.04%	**56.52%**	30.43%
Stuck	0	0	**100%**

10 sample outputs from the classifier are sent to the traffic congestion detection algorithm, which determines the current state of traffic. This takes a total of 123 seconds. As the previous state of the vehicle also plays a major role in determining the current traffic state, the last 50% samples of previous batch is also included along with the most recent five samples to determine the new state of traffic. Thus, we get new traffic states every 50 seconds.

An extensive set of experiments were carried out not only on private vehicles but also on public transports including buses and three-wheeled auto-rickshaws. The journeys were performed during both peak and non-peak hours. The results of one such bus journey is presented in Figure 5. This figure represents three plots each corresponding to different phones, Google Nexus 5, Google Nexus 5x, and Xiaomi Mi4, respectively. The x-axis represents the time in seconds and y-axis represents GPS speed in km/hr during the journey. The GPS speed is calculated for each 123 seconds of the journey by using the latitude and longitude to calculate the distance covered in the given time using Haversine formula. The three states *"moving," "congestion,"* and *"stuck"* are represented by green, orange, and red colors, respectively. In the plots most of the *"stuck"* states are encountered where GPS speed is below 10km/hr, most of the congested states are encountered where GPS speed is between 10–20km/hr, and above 20km/hr *"moving"* states are mostly encountered. In the plot for Nexus 5 and Xiaomi Mi4 there is no *"stuck"* state above 10km/hr, but in the plot for Nexus 5x few *"stuck"* states are encountered above 10km/hr, but maximum *"stuck"* states are encountered below 10km/hr. In between 10–20km/hr there is an intermix of *"moving"* and *"congestion"* states for all the three phones, but mostly *"congestion"* states are encountered between 10–20km/hr for all the three phones. This intermix is encountered due to the fact that in the speed range 10–20km/hr the vehicle could have experienced both *"still"* and *"motion"* activities and based on the number of *"still"* activity, *"moving"* and *"congestion"* states are determined. Above 20km/hr very few *"congestion"* states are encountered for Google Nexus 5 and Xiaomi Mi4, but most of the states are *"moving."* For Nexus 5x above 20km/hr there is one *"stuck"* state and some *"congestion"* states, but most of the encountered states are *"moving."*

Tables 2–4 show the detailed accuracies for Nexus 5, Nexus 5x, and Xiaomi Mi4 phones, respectively. It is clear that the accuracy of *"stuck"* state is very high, whereas the accuracy of *"moving"*

Table 3. Traffic State Accuracies for Nexus 5x, Corresponding to Figure 5(b)

	Moving (20+ km/hr)	Congestion (10–20km/hr)	Stuck (0–10km/hr)
Moving	**47.06%**	47.06%	5.88%
Congestion	31.81%	**54.54%**	13.63%
Stuck	3.33%	13.33%	**83.33%**

Table 4. Traffic State Accuracies for Xiaomi Mi4, Corresponding to Figure 5(c)

	Moving (20+ km/hr)	Congestion (10–20km/hr)	Stuck (0–10km/hr)
Moving	**42.42%**	48.48%	9.10%
Congestion	17.39%	**43.47%**	39.13%
Stuck	0	0	**100%**

Table 5. Altitude Changes Across Different Cities

City	Mean Altitude Change (meters)	Standard Deviation
Chandigarh	0.29	0.23
Mumbai	0.34	0.31
Mountain View (California)	0.28	0.48
New Delhi	0.27	0.22

and *"congestion"* is not very high. We will show in Section 6 how majority voting of crowdsourced data can help increase the accuracies of the *"moving," "congestion,"* and *"stuck"* states.

To identify other roads/cities where our approach of using barometer sensor could be implemented, we present altitude statistics for several cities in Table 5. We used the Google Elevation API to find the *altitude change* every 30 meters and calculate its mean and standard deviation. The Google Elevation API provides elevation data along the length of the path specified (set of latitude and longitude values), which in our case is sampled at every 30 meters. We covered several paths across all these cities, with a total path distance of 107km, and captured multiple elevation values at different resolutions.

From Table 5, we see that the mean altitude change for all the cities is in the range of 0.27–0.34 meter. The standard deviation is also similar across different cities, except for Mountain View, California. Our techniques for road traffic congestion and path estimation are based on the relative change in altitude in a period of a few seconds. As the mean altitude change for all the cities is similar to Chandigarh (the city where we primarily performed our experiments), we expect the proposed approach to work in other cities also.

5 PATH ESTIMATION

As shown in Figure 1, once a heavy traffic situation like *"stuck"* is detected, we get to know the coarse location of the user. Based on this location, we come to know the nearest intersection (A, as shown in Figure 1) to the user. A road stretch between two intersections is referred to as a road segment. The user could have traveled any of the possible road segments (AC, AB, AD, AG, CA, BA, DA, GA) or a part of them that are present in the vicinity of its nearest intersection (A) to reach its current location. Out of all these possible road segments, we find the road segment that the user traveled (BA) in real time to reach the congested area. This can help us to disseminate the traffic situation to other interested users in real time.

If the output of the traffic congestion detection algorithm is *"stuck,"* it means that the user did not travel any distance and hence there would not be any substantial changes in the barometer sensor values. Therefore, we should store the barometer sensor readings of the last *"moving"* state (classified by the traffic congestion detection algorithm) of the user and send these readings to the server to determine the most recent path the user traveled. The steps of the approach in brief are:

(1) On encountering a *"stuck"* state, the centralized server receives the current location of the user and the most recent batch of real-time barometer sensor readings of the last *"moving"* state.
(2) From the user's coarse location, the nearest intersection to that location is obtained as discussed in Section 3. The user could have traveled on any of the possible road segments in the vicinity of the intersection to reach its current location.
(3) To estimate the path the user traveled (during the *"moving"* state), real-time barometer sensor reading *(Q)* of *"t"* seconds from the user's phone is compared with all the road signatures *(R)* of *"m"* seconds present in the database corresponding to the intersection obtained using Dynamic Time Warping (DTW).

DTW is a time series alignment algorithm that tells how similar two signals are; namely, query *(Q)* and reference *(R)* sequence. Algorithm 2 describes the Dynamic Time Warping algorithm in detail. The algorithm receives Q and R signal as input of length *"t"* and *"m,"* respectively. The matrix $D = \{d_{i,j}\}$ serves as a measure of distance between signals Q and R, where its $(i, j)^{th}$ entry stores the Euclidian distance between the ith index of Q and jth index of R. DTW uses cost matrix $G = \{g_{i,j}\}$ to find the minimum DTW cost path between two time-dependent sequences using dynamic programming approach, where i varies from 1 to *"t"* and j varies from 1 to *"m"* [27]. Initially, the value of $g(1, 1)$ is set to $d(1, 1)$. DTW normalized distance $DNorm(Q, R)$ serves as a measure to tell how similar two sequences are and is calculated as $g_{t,m}/t$. The lower the DTW normalized distance, the more similar the two sequences are. Thus, the sequence with the lowest DTW normalized distance is the estimated path.

The time complexity of DTW is O(tm), where "t" and "m" are lengths of two time series. In our use case, "t" is the length of the series from the user's smartphone and "m" is the length of one road signature. The length of the time series sequences is approximately 120 seconds. When the "stuck" state is encountered, the coarse location of the user along with the barometer readings are sent to the server for path estimation. At the server, based on the user's coarse location, all the nearby streets are matched using DTW with the user's barometer reading. For the city of Chandigarh, India, we found that on an average around 40 street matchings were done to estimate the path of the user, considering all the possible scenarios like the user covering partial road segments to covering multiple road segments. These scenarios are described and evaluated in detail later in this section.

The time taken to compute 40 DTW street matchings on a 2GHz Intel Core i5 processor is 0.21 second. Even for corner case scenarios where the DTW matchings reach a large number—say, 10 times the average (40*10=400)—the time taken was 1.65 seconds. This further highlights the effectiveness of our approach and its ability to scale.

In our case, as these matchings are done on a server, we can run multiple street comparisons in parallel to further reduce the matching time. In terms of the space required to store the signatures

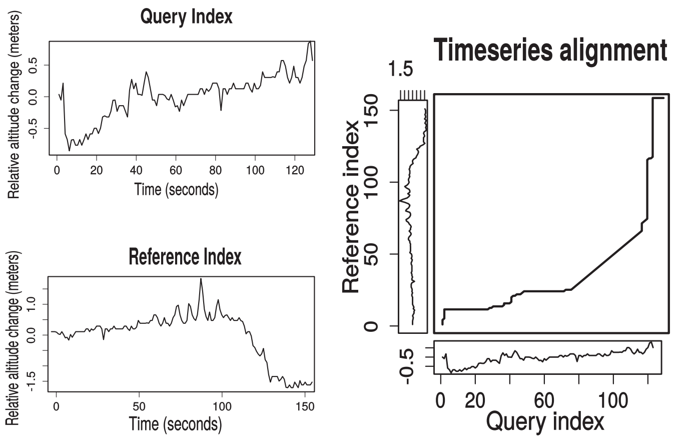

Fig. 6. Time series alignment of two sequences.

ALGORITHM 2: Dynamic Time Warping algorithm

```
1:  Input: Query Sequence(Q) of length t seconds, Reference Sequence(R) of length m seconds
2:  $D = \{d_{i,j}\} = |Q_i - R_j|^2$
3:  $G = \{g_{i,j}\} = 0$
4:  $g_{1,1} = d_{1,1}$
5:  for $i = 1$ To $t$ do
6:      $g_{i,1} = \sum_{k=1}^{i} d_{k,1}$
7:  end for
8:  for $j = 1$ To $m$ do
9:      $g_{1,j} = \sum_{k=1}^{j} d_{1,k}$
10: end for
11: for $i = 1$ To $t$ do
12:     for $j = 1$ To $m$ do
13:         $g_{i,j} = d_{i,j} + min(g_{i,j-1}, g_{i-1,j-1}, g_{i-1,j})$
14:     end for
15: end for
16: $DNorm(Q,R) = g_{t,m}/t$
17: return $DNorm(Q,R)$
```

in the database, a single road/path signature takes less than a 1KB. This means we can store roughly a million road signatures in about 1MB of memory. All the above statistics and conditions point to the fact that we do not need an overwhelmingly large data center and that path estimation can be done in a reasonable amount of time with reasonable computation resources.

Further, of the few hundred thousand of users using our app in a city, not all of them will be in a congested state at the same time. Of the few thousand of them who are in a congested state, their barometer readings are received by the server for path estimation. All of the requests are not processed at the same time instant, as these requests for path estimation are uniformly spread across 50 seconds (same as the update time interval of RoadSphygmo), since each user can turn on the application at any second, "s" of a minute, "m."

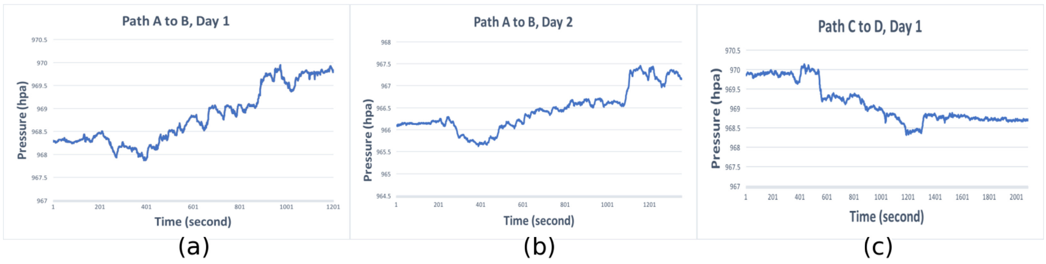

Fig. 7. Road Signatures on Different days.

Figure 6 contains a query sequence (real-time barometer sensor reading), reference sequence (road signature stored in database), and the time series alignment of these two sequences. The time series alignment gives us a measure of similarity between the two sequences, and the reference sequence that is most similar, i.e., which has the minimum DTW normalized distance to the query sequence, is the estimated path of the user.

We choose DTW for comparing the two road signatures because of the following advantages:

(1) Due to different length of the road segments around an intersection and varying user speeds, query and reference sequences are of different lengths. DTW is capable of comparing these signals of different lengths, hence it can compare the *"t"* seconds of user data with pre-stored road signatures that could be of different length.
(2) DTW can compare two road signatures of which one road signature is a subset of another road signature, therefore there is no need to store partial road segments in database.

Figures 7(a) and 7(b) show the road signature of the same road on two different days. The environmental factors such as temperature, wind speed, and so on may affect the pressure at a location. We observe that even though the absolute pressure for the same road for different days might be different, the signatures of the same road are similar across different days and very different from other road signatures (Figure 7 allows us to uniquely identify the path the user took using DTW). Due to the above advantages of DTW, the pattern observed by a user that has varying speeds can also be matched with a corresponding road segment for that user. Since the absolute pressure for the same road on different days might be different, as we saw in Figure 7, all the road signatures are obtained using the relative altitude change rather than using raw altitude values. The relative altitude values are calculated with respect to the first altitude value. So, even though the base pressure/altitude may be different for different journeys, the relative altitude change will be independent of this base pressure/altitude.

For different cities, the length of the road segments vary. Based on the average length of the road segments and average user speeds, we choose the value of *"t"* for a particular city. We evaluated the proposed method of path estimation in Chandigarh (India). Based on the fact that average length of the road segments in Chandigarh is 1km and considering average speed of the vehicle to be around 60km/hr, we choose t = 60 seconds.

We collected data in Chandigarh (India) covering an area of 24 sq km consisting of a total of 49 road segments to make the database of road signatures.

Multiple journeys were carried out over the same area of 24 sq km spanning over a period of three months on different days at different times (hence, different climatic conditions). Journeys were carried out in multiple vehicles having varying average speed and along the two directions of a road segment. During the course of the journeys to check the robustness of the barometer, different scenarios were tried out. The AC of the vehicle was turned on and off, the windows

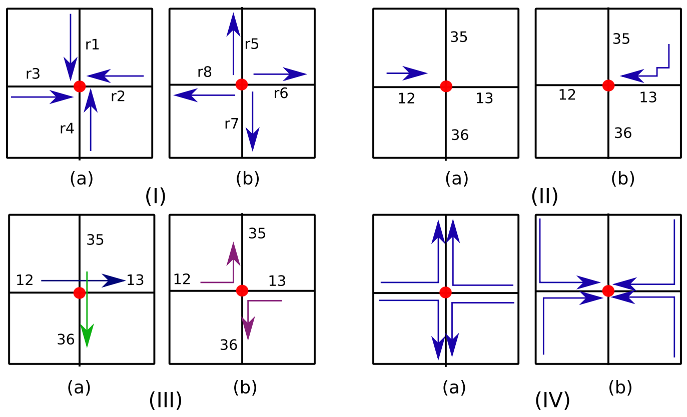

Fig. 8. Different cases of user travel.

Table 6. 8*8 Matrix of One Road Segment

	r1	r2	r3	r4	r5	r6	r7	r8
q1	**0.11**	7.92	0.30	6.97	1.54	3.89	1.09	5.83
q2	8.00	**0.10**	5.32	0.49	2.35	3.64	2.96	3.27
q3	0.29	5.38	**0.11**	4.48	1.11	1.84	0.92	3.57
q4	7.41	0.77	4.79	**0.07**	1.97	1.52	2.51	1.17
q5	1.74	2.77	1.04	2.07	**0.10**	0.51	0.31	1.04
q6	4.34	3.66	2.10	1.48	0.42	**0.09**	0.55	0.11
q7	1.06	3.21	0.77	2.41	0.37	0.63	**0.16**	1.32
q8	6.63	3.38	4.06	1.21	1.09	0.13	1.71	**0.07**

Comparing each road segment with all the possible 8 segments.

opened and closed, phones were kept in handbags and pockets of the user, and so on. All these scenarios did not affect the method for path estimation. For data collection, a total of 150km were covered in 5 hr with each of the multiple devices (Google Nexus 5, Google Nexus 5x, Xiaomi Mi3, Xiaomi Mi4).

Depending upon the speed and path followed by the user, the user will cover different distances, and hence different scenarios may arise, which are discussed below.

Scenario 1: One road segement. Considering only one **complete** road segment, we get 8 possible paths (4 where user is traveling towards the intersection and 4 away from it), as shown in Figure 8(I), that the user could have traveled to reach its current location.

To estimate the exact path, the user's real-time barometer values will be compared with each of the 8 road signatures corresponding to a particular intersection. We present detailed results of one such intersection case.

Table 6 shows the 8*8 matrix for the DTW normalized distances of one of the intersections. The column denotes the values as present in the database. The row markers represent the user's readings (ground truth). Each row shows that real-time user data of road segment q1 is compared against r1–r8 road signatures in the database. The minimum value indicates the estimated path. We observe that in case of q1 the value of r1 is minimum, so r1 is the estimated path of the user. Ideally, the minimum values should be along the diagonal of the matrix. The minimum values have been highlighted and all of these lie along the diagonal. Thus, we can conclude that the correct path the user took can be estimated with a high degree of confidence.

In all, we considered a total of 10 such road intersections. Each of these intersections with 8 road segments gives a total of 80 road segments. Of these 80 road segments, 72 were estimated correctly. This gives an accuracy of 90%.

Scenario 2: Subset of one road segment. When the user is traveling at 30–40km/hr, the distance covered is around 500–600 meters. Thus, the user just covers a subset of a complete road segment, as shown in the Figure 8(II)(a). We conducted multiple experiments using different phones and observed that the estimated path is the same complete major road segment that was a part of the user's journey and was covered partially.

However, since our database does not contain the inner, narrow, smaller roads inside a particular sector, there is a possibility that the 60-second readings obtained from the user may contain some portion of these inner roads and some portion of the major road segments that are present in the database, as shown in Figure 8(II)(b). In this case also, our experimental results show that the estimated path is of the same complete major road segment that was a part of the user's journey and was covered partially.

Scenario 3: Partial road segment composing two road segments. It is also possible that a user travels partial portions of two road segments in the duration of 60 seconds. These are called partial road segments. Let us consider an intersection shown as red dot in Figure 8(III). The nearby road segments are 12, 13, 35, 36. Two cases of partial road segments with 50%–50% and 25%–75% of the road segments are shown in Figure 8(III)(a).

The colored lines depict the actual path traveled by the user. The blue path consists of 50% of 12 and 50% of 13 road segments. When the blue road segment was compared with all the corresponding 8 road segments, i.e., 12, 13, 35, 36, and reverse direction of these road segments the estimated path was always either of the road segment from which this partial road segment is made of, i.e., 12 or 13.

In Figure 8(III)(a), the path denoted by the green line consists 25% of 35 road segments and 75% of 36 road segments. Figure 8(III)(b) shows perpendicular partial road segments in purple color. We conducted multiple experiments using different phones and observed that the estimated path was always either of the road segments from which this partial road segment was made of.

Scenario 4: Two complete road segments. If a user covers more than one road segment in 60 seconds, then the user must be traveling at a very high speed. To cover 1.5km, the user's speed should be 90km/hr, and to cover two complete road segments, the user's speed should be 120km/hr. Such driving conditions are obtained on highways where there is no chance of *"stuck"* state, so our system will not be triggered to estimate the path. In city roads such high speeds are rare and can be considered as corner cases. When we considered these corner cases, we found out that if in the database only 1 road segment is present and is compared with two road segments traveled by the user, then we do not get good results. To overcome this, we performed more experiments by enriching the database by adding two road segments as well.

For a given road intersection there are 26 possible two-road segments, some of which are shown in Figure 8(IV). The database therefore needs to be enriched with a total of 34 road segments (26 two-road and 8 one-road segments) for the given intersection. The user's real-time data of 60 seconds will be compared with each of the possible 34 road segments. Ideally, the minimum DTW normalized distance should be obtained for the same road segment in the database on which the user actually traveled.

For a particular intersection the user could have traveled any of these 34 road intersections to reach its current location. Thus, we need to compare the real-time user data with all the 34 road segments. Considering all the cases, we get a 34*34 matrix of DTW normalized distances. In 29 of these 34 cases, the path was estimated correctly, thus giving an accuracy of 85.29%.

Parameter values for different cities: The length of road segments can be less than or greater than 1 km in cities other than Chandigarh. In case the road segment is greater in length, the user will cover some part of the road segment in 60 seconds. The readings of this subset of road segment covered by the user will be compared against the single road segments in the database. This is same as a subset of road being compared against a complete road segment (scenario 2). It is also possible that the road segments can be of length less than 1km. So, a user can cover multiple road segments in the duration of 60 seconds. As proposed above, adding two road segments can cover this case (scenario 4). An alternative way would be to think: *Is it possible to just have single road segments in the database?* Yes, based on the average length of the road segments and speed of the vehicle time duration of the user's barometer sensor readings, *"t"* seconds used for comparison can be changed. Consider an example of a city where the average length of the road segments is 500 meters. Considering the average speed of the user to be around 60km/hr, the user will take roughly 30 seconds to cover the road segment. In this case, we can set *"t"* to 30 seconds. So, the database needs to have only single road segments of 500 meters length.

Robustness in stop-and-go traffic: When a vehicle is approaching the middle of a congested road segment, a driver often brakes a lot. Our activity recognition scenario takes 20 seconds of readings to classify it as "motion" or "still" state. The braking scenario will result in alternating outputs from the activity recognition module to be "motion" or "still." Our traffic congestion algorithm outputs a state considering barometer readings of over 2 minutes that have multiple motion and still states. Based on the frequency of motion and still states over a period of 2 minutes, the traffic congestion module will predict "moving," "congestion," or "stuck" states. If "stuck" is detected due to this frequent braking scenario, then our last "moving" state contains barometer values of 2 minutes, which is information-rich, and will fall in one of the four scenarios discussed above. Thus, our path estimation will work correctly.

6 IMPROVING ACCURACY WITH CROWDSOURCING

To use the crowdsourced data to improve the accuracy of the system by majority voting, we need to know which users are traveling together. We accomplish this by calculating the correlation values of raw altitude readings obtained from users' phones. The users who are close to one another will observe the same set of elevation changes along the stretch of the road. Hence, the altitude readings will be closely related to each other, and thus the correlation value should be high. The more the users are apart, the less will be the similarity of altitude changes they observe, which will lead to lower correlation values.

6.1 Determining Proximity of Users Using Correlation

To extensively study how correlation values depend on the degree of proximity of the users, we performed various experiments ranging from no overlap to complete overlap of the journey. We use 40 hr of our experimental data for correlation experiments. When the users took the same route throughout the journey, the correlation values were extremely high. When the journey of the users had no overlap, the correlation values were very low. In case of partial overlap, the correlation values drop compared to complete overlap but are higher than the case of no overlap. These observations are summed up in Table 7.

6.2 Improving System Accuracy

On encountering the *"stuck"* state, we infer the rough location of the user as explained earlier in system overview section. Thus, we can perform majority voting of only the users in same area. Our traffic congestion detection algorithm takes 123 seconds' worth of values (we obtain the 123-second interval based on having a sliding window batch of 10 activity states, as described in

Table 7. Correlation Results

Type of journey	Correlation value
Complete overlap	0.99
Partial overlap	0.85
No overlap	0.36

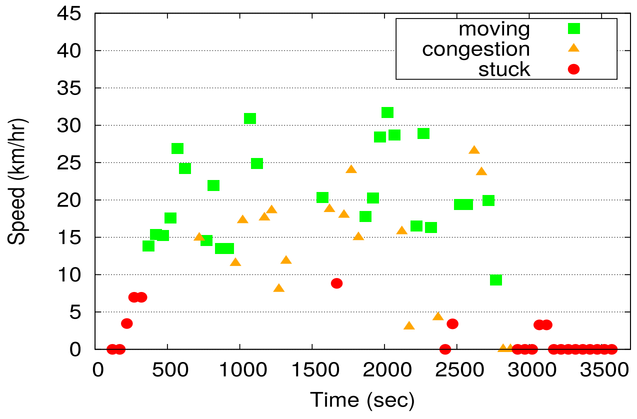

Fig. 9. Majority voting.

Table 8. Traffic State Determined by Majority Count

	Moving (20+ km/hr)	Congestion (10–20km/hr)	Stuck (0–10km/hr)
Moving	**80.37%**	16.82%	2.81%
Congestion	14.58%	**70.84%**	14.58%
Stuck	0	2.64%	**97.36%**

Section 4.2 and shown in Figure 4) to classify the traffic state. The correlation values are calculated for the same time interval (every 123 seconds, updated every 50 seconds) as in our traffic congestion detection algorithm.

The traffic congestion detection outputs of different phones (Nexus 5, Nexus 5x, and Xiaomi Mi4) in same vehicle are shown in Figure 5. To do majority voting, we choose the traffic state as final output that is in majority. However, if no traffic state is in majority, we do not choose any traffic state as final output. The results of majority voting for same is shown in Figure 9. In the plot the x-axis represents time in seconds, y-axis represents GPS speed in km/hr during the journey. The three states *"moving," "congestion,"* and *"stuck"* are represented by green, orange, and red colors, respectively. The GPS speed is calculated for each 123 seconds. In the plot there is no *"stuck"* state above 10km/hr. Most of the congested states are encountered when GPS speed is between 10–20km/hr, and most of the *"moving"* states are encountered when GPS speed is above 20km/hr. Thus, we can see that, using majority voting, we get better results as compared to individual phones.

Table 8 presents the accuracy of the traffic states after majority voting. We use all the data from buses, three-wheeled auto-rickshaws, and cabs to calculate the accuracy. Of all the *"moving"* states determined by traffic congestion algorithm, 80.37% of them are above 20km/hr, 70.84% of *"congestion"* states are in the speed range of 10–20km/hr, and 97.36% of *"stuck"* states are below

Table 9. Increase in Accuracy Due to Crowdsourcing

	Moving (20+ km/hr)	Congestion (10–20km/hr)	Stuck (0–10km/hr)
Nexus5	39.29%	25.33%	−2.64%
Nexus5x	70.78%	29.88%	16.83%
Xiaomi Mi4	89.46%	62.96%	−2.64%

10km/hr. However, we see that some of the *"congestion"* states determined by traffic congestion algorithm are above 20km/hr (14.58%) and some are even below 10km/hr (14.58%). This can be attributed to the fact that during *"congestion"* state, vehicles experience both *"motion"* and *"still"* activities. Along the same lines, our confidence regarding the congested state of the road segment increases with multiple users saying a particular road segment led them to a congested area and can help filter out outliers (rare cases where path estimation might predict the wrong path as the congested road segment).

Table 9 shows the impact crowdsourcing has over the ouptputs of individual mobile devices as shown in Tables 2, 3, and 4. Table 9 shows the increase in accuracy caused by crowdsourcing with respect to individual mobile phones. We notice a large increase in the accuracy for *"moving"* and *"congestion"* states. However, the increase in accuracy for *"stuck"* states is not very high. This is because the accuracy of *"stuck"* state is already very high, with Nexus 5 and Xiaomi Mi4 having 100% accuracy, as shown in Table 2 and 4, respectively. We see that for Nexus 5 and Xiaomi Mi4 the accuracy decreased slightly instead of increasing. This can be attributed to the fact that the *"stuck"* state accuracy for Nexus 5 and Xiaomi Mi4 phones is 100%. Whereas, the *"stuck"* state accuracy for Nexus 5x in Table 3 is 83.33%. So, if we take an average of the *"stuck"* state across all the three different phones, the accuracy would be less than 100%. But for *"moving"* and *"congestion"* states, the increase in accuracy across all the different phones is in the range 25.33%–89.46%. We suspect that with more mobile devices available for crowdsourcing these numbers would be even better. This gives us an idea that, with crowdsourcing, the accuracies of predicting the state of traffic can be increased significantly.

6.3 Other Interesting Applications

Apart from majority voting, we can also use the correlation values for some other applications, which are explained below:

Determine when users part ways: On careful observation of correlation values at regular intervals, we can also determine when the users part ways during their course of the journey. To do so, we calculate the correlation values for every 123 seconds (see reasoning below) with an update interval of 50 seconds (just as in the congestion detection algorithm). We calculate the correlation value for each window. As the users part ways the correlation value will gradually begin to drop. As correlation value drops below a certain threshold (say, 0.8), we can say the users parted ways. Table 10 shows correlation values for two users for the sliding window. As the correlation value in windows number 1 and 2 are high (greater than our threshold of 0.80), the vehicles are traveling together, but in window number 3 the correlation value becomes 0.56, which is less than our threshold. We can now trigger GPS/GSM to get the location where the users parted ways. We should be careful and avoid large window sizes, as users can part ways and travel way further along their respective different routes. Hence, we will not be able to determine the exact location confidently at the end of window in which the users parted ways.

Boarding and Deboarding of a vehicle: We expanded the use of the above approach to know when the user boarded and deboarded a vehicle. We consider one mobile phone to be present in the

Table 10. Correlation Values in Sliding Windows to Determine When Users Parted Ways

Window number	Starting time (sec)	End time (sec)	Correlation value
1	1	123	0.97
2	51	173	0.93
3	101	223	0.56
4	151	273	0.72

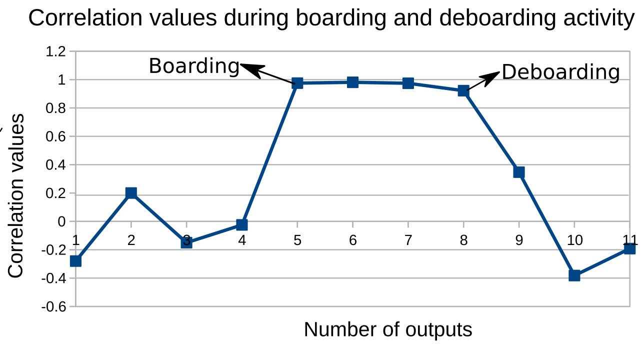

Fig. 10. Detecting boarding and deboarding using correlation.

vehicle **(driver's mobile phone)** and another with the user. We performed multiple experiments related to boarding and deboarding. Figure 10 presents the results of one such experiment. The x-axis shows the output number of the correlation values. The y-axis shows the correlation values. We see that when the user boards the vehicle, the correlation value rises and becomes close to 1. As long as the user is in the vehicle, the correlation value will remain close to 1 (users in same vehicle). The deboarding activity can be detected when the correlation value drops. As we have to find the location where boarding and deboarding activities took place, it becomes important that the duration of altitude values for which we calculate the correlation is not large, else we will not get the exact location.

Occupancy rate of vehicle: This also brings us to an interesting case where we can determine the occupancy rate of the vehicle using the above results. When we observe that multiple users are having high correlation values, it could also be the case if the two users were in separate vehicles but moving alongside one another. But we have seen that maintaining very high correlation values when in different vehicles is not possible in real-time traffic. Hence, to determine the occupancy rate of the vehicles we will observe the correlation values of the users who are contenders to be traveling in the same vehicle. If the correlation value of the single user as compared to a set of other users drops to a lower value (<0.97), then we can say that particular user was not traveling in the same vehicle as other users. Hence, observing some windows of barometer readings will help us determine the occupancy rate of the vehicle, too.

For all these applications, we need to be sure that the vehicle is not in a *"stuck"* state, because then the altitude will not change and the correlation values will be high. Thus, we will not trigger the correlation-related events such as boarding/deboarding or users parting ways when the user is continuously in the *"stuck"* state. For all of the above experiments, we have used a window of 123 seconds with an update interval interval of 50 seconds. This helps us in using the same set of readings for all the correlation-related applications that use traffic congestion detection.

7 CONCLUSION AND FUTURE WORK

We presented BaroSense, a system that uses the barometer sensor present in most users' mobile phones to detect the state of traffic into *"moving," "congestion,"* and *"stuck"* states. When *"stuck"* state is encountered, we estimate the path the user traveled to reach the congested area. We achieved an accuracy of 80.37%, 70.4%, 97.36% on *"moving," "congestion,"* and *"stuck"* states, respectively. Considering different cases of user travel, we estimated the correct path in all the cases. We found that users traveling together have a high degree of correlation. Thus, we can do majority voting of crowdsourced data of the users who are traveling together, which helps in increasing the accuracy of the BaroSense system. Using the concept of correlation, we can detect events such as boarding and deboarding of a vehicle. Since barometer is a passive sensor, orientation-independent, and is robust in different environmental conditions and consumes much less power than other sensors, it can be effectively used to provide a low-cost approach for detecting traffic congestion and estimating the path of the user.

ACKNOWLEDGMENTS

We would like to thank Simran Jain and Samriddhi Singla, B.E. CSE students at UIET, Panjab University for their immense support throughout this project.

REFERENCES

[1] 2019. Accelerometer datasheet. Retrieved from: https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMA400-DS000.pdf.

[2] 2019. Accelerometer power. Retrieved from: https://www.electronicspecifier.com/sensors/accelerometer-boasts-ultra-low-power-consumption.

[3] 2019. Barometer power. Retrieved from: https://www.electronicspecifier.com/sensors/barometric-pressure-sensor-provides-low-current-consumption.

[4] Robert Bogue. 2013. Recent developments in MEMS sensors: A review of applications, markets and technologies. *Sensor Rev.* 33, 4 (2013), 300–304.

[5] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.* 2, 3 (2011), 27.

[6] Sing Cheung, Sinem Coleri, Baris Dundar, Sumitra Ganesh, Chin-Woo Tan, and Pravin Varaiya. 2005. Traffic measurement and vehicle classification with single magnetic sensor. *Transport. Res. Rec.: J. Transport. Res. Board* 1917 (2005), 173–181.

[7] Anuj Dimri, Harsimran Singh, Naveen Aggarwal, Bhaskaran Raman, Diyva Bansal, and K. K. Ramakrishnan. 2016. RoadSphygmo: Using barometer for traffic congestion detection. In *Proceedings of the 8th International Conference on Communication Systems and Networks (COMSNETS'16).* IEEE, 1–8.

[8] Globalsat. 2014. Globalsat em-506 GPS module. Retrieved from: https://www.globalsat.com.tw/ftp/download/EM-506RE%20Hardware%20Data%20Sheet%20V1.7.pdf.

[9] Jun Han, Emmanuel Owusu, Le T. Nguyen, Adrian Perrig, and Joy Zhang. 2012. Accomplice: Location inference using accelerometers on smartphones. In *Proceedings of the 4th International Conference on Communication Systems and Networks (COMSNETS'12).* IEEE, 1–9.

[10] Juan C. Herrera, Daniel B. Work, Ryan Herring, Xuegang Jeff Ban, Quinn Jacobson, and Alexandre M. Bayen. 2010. Evaluation of traffic data obtained via GPS-enabled mobile phones: The Mobile Century field experiment. *Transport. Res. Part C: Emerg. Technol.* 18, 4 (2010), 568–583.

[11] Bo-Jhang Ho, Paul Martin, Prashanth Swaminathan, and Mani Srivastava. 2015. From pressure to path: Barometer-based vehicle tracking. In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments.* ACM, 65–74.

[12] Satoshi Hyuga, Masaki Ito, Masayuki Iwai, and Kaoru Sezaki. 2015. Estimate a user's location using smartphone's barometer on a subway. In *Proceedings of the 5th International Workshop on Mobile Entity Localization and Tracking in GPS-less Environments.* ACM, 2.

[13] Andreas Janecek, Karin A. Hummel, Danilo Valerio, Fabio Ricciato, and Helmut Hlavacs. 2012. Cellular data meet vehicular traffic theory: Location area updates and cell transitions for travel time estimation. In *Proceedings of the ACM Conference on Ubiquitous Computing.* ACM, 361–370.

[14] G. Lammel, J. Gutmann, L. Marti, and M. Dobler. 2009. Indoor navigation with MEMS sensors. *Procedia Chem.* 1, 1 (2009), 532–535.

[15] Mingqi Lv, Ling Chen, Xiaojie Wu, and Gencai Chen. 2015. A road congestion detection system using undedicated mobile phones. *IEEE Trans. Intell. Transport. Syst.* 16, 6 (2015), 3060–3072.

[16] Greg Milette and Adam Stroud. 2012. *Professional Android Sensor Programming.* John Wiley & Sons.

[17] Brendan Tran Morris and Mohan Manubhai Trivedi. 2008. A survey of vision-based trajectory learning and analysis for surveillance. *IEEE Trans. Circ. Syst. Vid.Technol.* 18, 8 (2008), 1114–1127.

[18] Kartik Muralidharan, Azeem Javed Khan, Archan Misra, Rajesh Krishna Balan, and Sharad Agarwal. 2014. Barometric phone sensors: More hype than hope! In *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications.* ACM, 12.

[19] Jeongyeup Paek, Joongheon Kim, and Ramesh Govindan. 2010. Energy-efficient rate-adaptive GPS-based positioning for smartphones. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services.* ACM, 299–314.

[20] Kartik Sankaran, Minhui Zhu, Xiang Fa Guo, Akkihebbal L. Ananda, Mun Choon Chan, and Li-Shiuan Peh. 2014. Using mobile phone barometer for low-power transportation context detection. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems.* ACM, 191–205.

[21] Rijurekha Sen, Abhinav Maurya, Bhaskaran Raman, Rupesh Mehta, Ramakrishnan Kalyanaraman, Nagamanoj Vankadhara, Swaroop Roy, and Prashima Sharma. 2012. Kyun queue: A sensor network system to monitor road traffic queues. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems.* ACM, 127–140.

[22] Bosch Sensortec. 2015. Digital pressure sensor. Document number: BST-BME280-DS001-10. Revision_1.1 May 7th (2015). Retrieved from https://raw.githubusercontent.com/SeeedDocument/Grove-Barometer_Sensor-BME280/master/res/Grove-Barometer_Sensor-BME280-.pdf.

[23] Shuming Tang and Fei-Yue Wang. 2006. A PCI-based evaluation method for level of services for traffic operational systems. *IEEE Trans. Intell. Transport. Syst.* 7, 4 (2006), 494–499.

[24] Sha Tao, Vasileios Manolopoulos, Saul Rodriguez, Ana Rusu, et al. 2012. Real-time urban traffic state estimation with A-GPS mobile phones as probes. *J. Transport. Technol.* 2, 1 (2012), 22.

[25] Suttipong Thajchayapong, Wasan Pattara-Atikom, Noppadol Chadil, and Chaichana Mitrpant. 2006. Enhanced detection of road traffic congestion areas using cell dwell times. In *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC'06).* IEEE, 1084–1089.

[26] Veena Tyagi, Shivkumar Kalyanaraman, and Raghuram Krishnapuram. 2012. Vehicular traffic density state estimation based on cumulative road acoustics. *IEEE Trans. Intell. Transport. Syst.* 13, 3 (2012), 1156–1166.

[27] Wikipedia. 2016. Dynamic time warping—Wikipedia, The Free Encyclopedia. Retrieved from: https://en.wikipedia.org/w/index.php?title=Dynamic_time_warping&oldid=729379715.

[28] Myounggyu Won, Shaohu Zhang, Appala Chekuri, and Sang H. Son. 2016. Enabling energy-efficient driving route detection using built-in smartphone barometer sensor. In *Proceedings of the 19th IEEE International Conference on Intelligent Transportation Systems (ITSC'16).* IEEE, 2378–2385.

[29] Jungkeun Yoon, Brian Noble, and Mingyan Liu. 2007. Surface street traffic estimation. In *Proceedings of the 5th International Conference on Mobile Systems, Applications, and Services.* ACM, 220–232.

[30] Jieying Zhang, Ezzaldeen Edwan, Lunchuan Zhou, Wennan Chai, and Otmar Loffeld. 2012. Performance investigation of barometer aided GPS/MEMS-IMU integration. In *Proceedings of the IEEE/ION Position Location and Navigation Symposium (PLANS'12).* IEEE, 598–604.

Received November 2017; revised July 2019; accepted September 2019