Using Mobile Phone Barometer for Low-Power Transportation Context Detection

Kartik Sankaran[†], Minhui Zhu[†], Xiang Fa Guo[†], Akkihebbal L. Ananda[†],
Mun Choon Chan[†], and Li-Shiuan Peh[‡]
[†]School of Computing, National University of Singapore
[‡]Dept. of EECS, Massachusetts Institute of Technology
[†]{kartiks,zhuminh,xiangfa,ananda,chanmc}@comp.nus.edu.sg, [‡]peh@csail.mit.edu

Abstract

Accelerometer is the predominant sensor used for low-power context detection on smartphones. Although low-power, accelerometer is orientation and position-dependent, requires a high sampling rate, and subsequently complex processing and training to achieve good accuracy. We present an alternative approach for context detection using only the smartphone's barometer, a relatively new sensor now present in an increasing number of devices. The barometer is independent of phone position and orientation. Using a low sampling rate of 1 Hz, and simple processing based on intuitive logic, we demonstrate that it is possible to use the barometer for detecting the basic user activities of *IDLE*, *WALKING*, and *VEHICLE* at extremely low-power. We evaluate our approach using 47 hours of real-world transportation traces from 3 countries and 13 individuals, as well as more than 900 km of elevation data pulled from Google Maps from 5 cities, comparing power and accuracy to Google's accelerometer-based Activity Recognition algorithm, and to Future Urban Mobility Survey's (FMS) GPS-accelerometer server-based application. Our barometer-based approach uses 32 mW lower power compared to Google, and has comparable accuracy to both Google and FMS. This is the first paper that uses only the barometer for context detection.

Categories and Subject Descriptors

I.5.4 [**Pattern Recognition**]: Applications—*Signal processing*

General Terms

Algorithms, Experimentation

Keywords

Context detection; Mobile sensing; Barometer

SenSys'14, November 3–5, 2014, Memphis, TN, USA.

ACM 978-1-4503-3143-2/14/11 ...$15.00
http://dx.doi.org/10.1145/2668332.2668343

1 Introduction

With smartphones now reaching the computational power of personal computers, they are expected to behave intelligently: they should silently understand what the user is doing, help in ongoing or future tasks, and adapt accordingly. Google Now (July 2012) and Siri (Oct 2011), for example, behave as intelligent personal assistants. Cover [1], an Android application released in Oct 2013, automatically adapts the applications displayed on the lockscreen based on whether the user is at home/work/travelling. Google utilizes the user's activity and movement to improve the quality of location readings [2].

A key ingredient of such intelligent smartphone behaviour is context-awareness. The phone needs to continuously understand what the user is doing. Context is typically derived from the multitude of sensors on the phone. Since the phone's battery life-time is critical, context-detection algorithms must run at extremely low-power. In this regard, Apple and Google have taken the first steps forward to reducing power consumed for walking and step detection, by introducing the M7 co-processor [3] (Sept 2013) and step counter [4] (Oct 2013) to offload sensor processing from the main CPU when the phone is asleep.

Transportation-mode detection is a special case of context-awareness where the phone automatically understands the user's daily commute. Such awareness is immensely useful for maintaining activity diaries, conducting surveys, and urban planning. For example, Moves [5], the first to make use of Google's Activity Recognition API [6], is a popular application that maintains an activity diary for the user, and was featured in the Google I/O session in May 2013 when the API was introduced.

Being one of the lowest-power sensors available on the phone, accelerometer is the predominantly used sensor in transportation-mode detection [7]. It can detect acceleration in the phone's 3 axial directions. Using supervised machine learning and features extracted from the accelerometer readings, user activity can be classified as *IDLE*, *WALKING* or *VEHICLE*.

Although accelerometer is low-power, its readings are phone orientation and position-dependent, as well as user and vehicle-dependent. The machine learning algorithm needs to be trained for all these different possibilities. To detect walking and vehicle activities accurately, sampling rate

is typically 10 Hz and above. The high sampling rate, 3 axial directions, and position dependence make the classification complicated and increases power consumption (Section 6.4).

In this paper, we present an alternative approach to context-detection using only the barometer, a sensor now found in an increasing number of devices today[1], which measures air pressure. Pressure can in turn be translated to altitude (height). Barometers were initially introduced on Android phones to reduce the delay of the GPS fix by providing the z co-ordinate (altitude). Since the MEMS sensor is sensitive enough to measure even a 1 metre change in height, they are also used for floor-change detection, and can differentiate between travelling on stairs/elevator. Some applications are attempting to crowd-source pressure data for weather forecasting [8, 9]. Fitness applications use change in height to better estimate calorie consumption.

We present the first paper that uses only the barometer for transportation context-detection. The barometer is inherently orientation and position-independent. Using a low sampling rate of 1 Hz, coupled with the new sensor-batching hardware, and using relatively simple processing based on intuitive logic, we demonstrate that the barometer can be used for basic context detection of the user activities *IDLE*, *WALKING*, and *VEHICLE* at extremely low power. These states are sufficient to characterise typical transportation context, and can act as a trigger for other higher-power finer-granularity classification of vehicular modes.

We evaluate our approach using 47 hours of transportation traces from 3 countries and 13 individuals. We compare the accuracy and power consumption to two other approaches: Google's accelerometer-based Activity Recognition algorithm [6], which runs at low power, and Future Urban Mobility Survey's (FMS) GPS and accelerometer server-based approach [10, 11], an app developed by the Singapore-MIT Alliance for Research and Technology (SMART), in conjunction with the Singapore government's Household Interview Travel Survey (HITS). FMS is the first smartphone survey app to have been field-tested on a large deployment of over 1000+ users in Boston and Singapore, designed as an alternative to traditional surveys done via in-person interviews in HITS.

We chose FMS and Google as baselines due to their wide deployment and usage. In our evaluation, we find that our barometer-based approach consumes 32 mW lower power in comparison to Google, and has comparable accuracy to both Google as well as FMS.

In addition to real-world trace data, we have also evaluated our algorithm using over 900 km (30,000 data points) of elevation data available on Google Maps from 5 cities. The accuracy results from map data are similar to real-world trace data, and provide convincing evidence that sufficient elevation changes do occur in practice for barometer to detect user context.

We have implemented our approach on Android. It runs in real-time on the phone locally, without requiring internet connection.

[1]Nexus 4, Nexus 5, Galaxy Nexus, Galaxy S3/4/5, Galaxy Note 1/2/3, and many more

The rest of the paper is organised as follows: Section 2 discusses related work. Section 3 describes the motivation of using the barometer for context detection. A brief background is given in Section 4, and the context-detection methodology is described in Section 5. Section 6 evaluates the barometer approach. This is followed by a discussion in Section 7, and Section 8 concludes the paper.

2 Related Work

Research has been done in context-detection using sensors for several years, differing in the type of user activities detected, sensors used, and classification techniques. An extensive survey is presented in [7]. Accelerometer is the predominant sensor used, with 28 out of 36 papers listed using it. Other commonly used sensors include GPS, Cellular, and WiFi. In the following, we focus on prior work using these sensors, describing their limitations, while section 3 discusses how the barometer overcomes these limitations.

- **GPS:** GPS is an extremely useful sensor for activity detection, since it provides the physical location of the user. A series of GPS readings can be analysed to calculate speed and bearing, typically used in combination with accelerometer features to achieve higher classification accuracy. Reddy et al. [12] and Ryder et al. [13] use GPS in conjunction with accelerometer to detect the modes idle/walking/running/bike/vehicle, and observe that GPS contributes to an increase in accuracy of about 10%. The Future Urban Mobility Survey (FMS) application [10, 11] uses GPS and accelerometer to detect the modes walking/car/bus/MRT/bike, observing an increase of 27% in accuracy over just using an accelerometer sampled at 10 Hz. In contrast, Zheng et al. [14] uses solely GPS data to detect the modes walking/driving/bus/bike, by extracting additional features such as heading change rate, stop rate, and velocity change rate. The trade-off in GPS accuracy is its high power usage and low coverage indoors, underground, and in urban canyons. To reduce power usage, GPS is turned on adaptively rather than periodically, using accelerometer and/or cell tower change as a trigger. However, the power usage remains high outdoors, where charging sockets are typically unavailable.
- **Cellular and WiFi:** Movement can also be detected using change in cellular and WiFi signals. Anderson and Muller [15] use fluctuation in cellular signal strength to figure out if the user is stationary/walking/driving, as does Sohn et al. [16]. Using signal strength is challenging since it can change unpredictably even when the user is stationary. Nawaz et al. [17] use a more robust WiFi beacon reception ratio as opposed to signal strength, to detect if a user has parked or is driving a car. An algorithm called BeaconPrint [18] uses cellular/WiFi beacon IDs to detect movement, without requiring any signal strength information. Since WiFi has shorter range and the network deployment is denser, change in signal is observed faster than change in cellular signal. However, WiFi-based techniques work only in urban areas with dense WiFi access points. Cellular

has better coverage, but cell size can vary significantly, making cellular-based detection difficult to generalize.

- **Accelerometer:** The lowest-power and most predominant sensor used for context detection is the accelerometer. Features extracted from accelerometer data are input to a supervised machine learning algorithm, which classifies the user activities. Due to classifier complexity, majority of prior work perform classification offline [19] instead of in real-time. Reddy et al. [12] implement their classifier on Nokia N95 phones, while performing the training offline, as does [20]. To avoid orientation problems, orientation-independent features (such as combined magnitude) can be used [19]. However, extensive training is still required to account for user and position dependence. Unlike Cellular and WiFi, accelerometer is capable of fine-grained classification of vehicular modes. Hemminki et al. [21] implement a three-stage classifier on Android to detect travelling on bus/train/metro/tram/car, at a power consumption of 85 mW (excluding base-power consumption). Our barometer-based approach can be used as a low-power trigger for higher-power finer-grained vehicular classification.
- **Barometer:** Barometer has been used for aiding GPS [22], the reason for its introduction into Android smartphones. Tanigawa et al. [23] uses barometer as an aid in removing accelerometer drift. Due to its excellent relative accuracy, barometer has been used for floor-change detection [24, 25]. Stairs and elevator can be easily distinguished using vertical speed thresholds. [24] uses a sampling rate of 1 Hz, similar to our paper. A higher sampling rate of 15 and 25 Hz can be used to reduce noise [25]. However, since newer barometer chips support internal hardware smoothing, a high sampling rate is no longer required. So far, barometer has been used only as an aid to other sensors. To the best of our knowledge, no prior work has used only barometer for detection of the modes idle/walking/vehicle.

3 Motivation

Prior work have used multiple sensors including GPS, Cellular, WiFi, and accelerometer for context detection. Then why it is advantageous to use barometer for the same purpose?

Although several sensors are available, each has its own set of limitations in low-power activity detection, listed in Table 1. GPS consumes high power, and has poor coverage indoors and underground. In contrast, the barometer sensor is one of the lowest powered sensors on the phone, and is available for use everywhere. WiFi and cellular-based approaches are better than GPS power-wise, but do not work without sufficient density of access points and cell towers. The barometer, on the other hand, is not dependent on any external infrastructure.

Accelerometer, like barometer, is low-power, and not dependent on external infrastructure. Consequently, it has become popular for context-detection. However, even accelerometer has drawbacks. By nature, accelerometer data is dependent on the phone's position (is the phone in a bag, pocket, or hand), and its orientation. Additionally, the readings vary from user to user, and vehicle to vehicle. Every user handles a phone differently, and different vehicles may produce different vibrations in the phone while moving. These dependencies can be offset by using orientation-independent features and training the machine learning algorithm with each dependency case. However, addressing these dependencies adds to the cost and complexity of the system, increasing power consumption. Majority of prior accelerometer work implement the classification offline instead of on the phone, concentrating on accuracy but neglecting power.

As we demonstrate later in this paper, barometer is inherently position-independent, requires simple processing and only minor calibration based on the terrain, overcoming all the drawbacks of accelerometer.

To summarize, barometer has the following advantages:

- **Position-independence:** Barometer measures air pressure, and is inherently position-independent, as long as the phone is not kept in an air-tight environment.
- **Simpler calibration:** Accelerometer depends on the position, orientation, user, and vehicle, and requires sufficient training to work well in all cases. Barometer reduces dependencies drastically: it only requires calibration of a few parameters using the overall characteristics of the terrain of the land, which remains relatively unchanged over time.
- **Better *WAIT* detection:** In transportation applications, one of the important aspects of the journey is the waiting time. While accelerometer is excellent for detecting when the phone is stationary, it faces problem when the user fiddles or makes minor movements with the phone, triggering false positives. This is especially problematic when accelerometer is used to trigger higher power sensors like GPS. Barometer, unaffected by phone movements, yields fewer false positives compared to accelerometer for user movement (Section 6.1.3 compares the accuracy of accelerometer and barometer for the *WAIT* state).
- **Lower-power:** Barometer's lower sampling rate and simpler processing reduce power consumption. Table 2 shows how the sampling rate can affect power usage (A higher sampling rate causes the sensor driver to be woken up more frequently). Sampling accelerometer at 20 Hz increases base power consumption by 112%, while sampling barometer at 1 Hz increases it only by 2% (For more details on the measurement settings, please see Section 6.4). Note that actual sampling rate is typically higher than what is specified to the Android Sensor Manager (for example, Galaxy S3 returns barometer values at 5 Hz rather than 1 Hz).

Accelerometer data in 3 axial directions is both a boon and a bane. It provides more information, but complicates processing. This paper ultimately tries to answer the question: Is one-dimensional height data more useful than three-

Table 1: Limitations of existing sensors for low-power activity detection

Sensor	Power	Limitations	Barometer advantage
GPS	Very high	Lack of indoor/underground coverage High power usage	Usable everywhere Ultra-low-power
WiFi/Cellular	High/Moderate	Requires dense access points/cellular towers	No external infrastructure
Accelerometer	Low	Extensive training required Classification complexity Position dependence	Simple calibration based on terrain Simple processing Inherently position independent

Table 2: Power Usage at different sampling rates (Galaxy S3, screen and data off, no sensor data processing. Accl and barometer power include CPU awake power. Note that actual sampling rate is higher than what is specified to the Android sensor manager)

	Power (mW)	Increase over base power
Accl (20 Hz)	230	112%
Accl (10 Hz)	180	67%
Accl (2 Hz)	164	51%
Baro (1 Hz)	110	2%
CPU Awake (base)	108	x
CPU Asleep	25	x

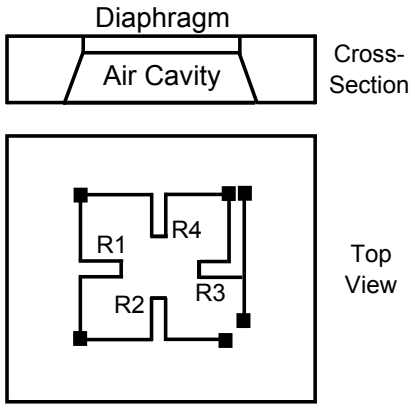


Figure 1: Simplified cross-section of MEMS piezoresistive barometer (adapted from [27])

dimensional accelerometer data? In other words, can more be done with less data?

4 Background

In order to better understand the barometer sensor characteristics, strengths, and sources of error, it is constructive to look at the MEMS barometer at a deeper level.

Popular MEMS barometers are of the piezoresistive type. They consist of a thin diaphragm over a small air cavity of near vacuum pressure [26]. Piezoresistors are arranged on the diaphragm in the form of a wheatstone bridge circuit (Figure 1). The external atmosphere exerts a pressure over the diaphragm, causing it to get depressed into the air cavity. This deflection causes a change in resistance of the piezoresistors, which in turn changes the voltage output of the wheatstone bridge. Using two or more calibration points, the change in voltage can be translated into corresponding air pressure values in millibar. The entire MEMS package is extremely small (for example, Bosch's BMP280 barometer on Nexus 5 measures 2 x 2.5 x 0.95 mm).

Possible sources of error in barometer are as follows:

- **Vibration:** One would expect that a phone movement or vibration would cause a major deflection in the diaphragm, leading to unpredictable pressure values. However, this is not the case. We conducted an experiment where we vibrated a Nexus 4 phone in the phone's 3 axial directions (one by one) using strong approximately 1 cm oscillations. However, this did not cause a change in the noise value (1 metre without smoothing), nor cause outliers. This is perhaps due to the small size of the diaphragm (in contrast, the MEMS accelerometer uses rods in order to amplify vibrations).
- **Temperature:** A temperature above or below room temperature (25 deg) causes a change in the resistance of the piezoresistors, leading to errors. Earlier barometer chips used a thermistor to compensate for this temperature error. Current chips contain a temperature sensor bundled into the package. The driver reads both pressure as well as temperature, and compensates for the error in software. Since the errors are usually second order or higher, a quadratic compensation is more effective than a linear compensation [28]. Since the barometer driver is part of the Android Open Source Project, we can check which phones perform the necessary temperature compensation (Table 3). Our experiments with Galaxy S3 show us that the error caused by using a linear instead of a quadratic correction is small, and does not affect our context-detection algorithm.
- **Installation bias and aging drift:** Installation bias (offset caused by soldering) is taken care of at the end of the phone's production line. Aging drift, which causes a drift in absolute pressure values as the barometer chip grows older, is in the order of months, and does not affect our algorithm which runs at a small time scale of a few minutes.
- **Weather drift:** Change in weather can cause a change in air pressure, and consequently a change in calculated height, even when the phone is still. Typical weather

Table 3: Summary of barometer chips on popular phones

Baro Chip	Phones	Temp Sensor	Temp Compensation	Oversampling	Noise filter
LPS331AP (STM)	Galaxy S3	Yes	Linear (on chip)	Yes	No
BMP180 (Bosch)	Galaxy Nexus/S4, Nexus 4	Yes	Quadratic (in driver)	Yes	No
BMP280 (Bosch)	Nexus 5	Yes	Quadratic (in driver)	Yes	Yes

drift is a few metres in an hour, but intense storms can cause a drift of 3 to 4 metres even in 10 minutes. We discuss and evaluate weather effects in more detail in Section 6.1.2.

- **Sunlight and wind:** The diaphragm and resistors, if exposed, will be affected by sunlight and wind. However, the barometer is protected under the phone's outer case from direct light. The MEMS package contains only a tiny air hole to capture air pressure, protecting it from wind. This matches our observations in windy weather (section 6.1.2).

Barometer chips come with the capability to internally oversample and smooth pressure values to reduce noise. Our inspection of the driver code tell us that the chips are already configured at optimized settings. Table 3 summarises the chips found on popular phones.

Air pressure (in millibar) can be translated into height (in metres) above sea level. The absolute accuracy of the height depends on the sea level reference pressure used, and the time of the day. In other words, for the same reference level, the barometer can very well report significantly different altitudes at different times of the day. To get an accurate estimation of altitude, the mean sea level pressure for the phone's region needs to be fetched from a local weather website at that time of the day, and used as a reference.

However, our context-detection algorithm does not require absolute accuracy, but rather good relative accuracy. Barometer chips on newer phones are sensitive enough to measure a change in height of even 1 metre, the reason why it is so useful for floor-change detection. The barometer's good relative accuracy is a strength exploited by our algorithm. Note that the height resolution is limited by the noise, which is approximately 1 metre without filtering. Nexus 5's chip, which performs internal filtering, has a lower noise value than other chips.

The study in [25] tests whether the change in height (i.e. relative accuracy) varies on different phones. Although different phones can report different absolute height values, the change in height values on different phones while moving are in sync.

In summary, by looking deeper into the MEMS barometer, we find that the main source of height error is weather drift. We will evaluate the impact of weather drift in section 6.1.2.

5 Methodology

In this section, we describe how exactly we use barometer to detect the states of *IDLE*, *WALKING* and *VEHICLE*.

5.1 Activity Definitions

Before describing our methodology, we need to first define the meaning of each state to avoid ambiguity. The state *VEHICLE* includes both motorised and non-motorised vehicles (including cycling). The state *IDLE* is not as strict as the typical definition in accelerometer-based works, since barometer is unaffected by hand movements. If a user moves around in the same room or floor of a building, we still consider it as an *IDLE* state. We argue that in the context of transportation, such movements should be clearly differentiated from the *WALKING* and *VEHICLE* states since these movements include important transportation context such as waiting at bus stops, taxi stands and subway platforms. This definition also yields fewer false positives compared to accelerometer when movement is used to trigger high power sensors such as GPS.

The states *IDLE*, *WALKING*, and *VEHICLE* are sufficient to characterise typical transportation context. Several popular applications such as Cover [1] and Moves [5] already make use of these fundamental three states. The *VEHICLE* state can also act as a trigger for other higher-power finer-granularity classification of vehicular modes. Similarly, additional sensors can be utilized to differentiate between stationary and waiting for transport.

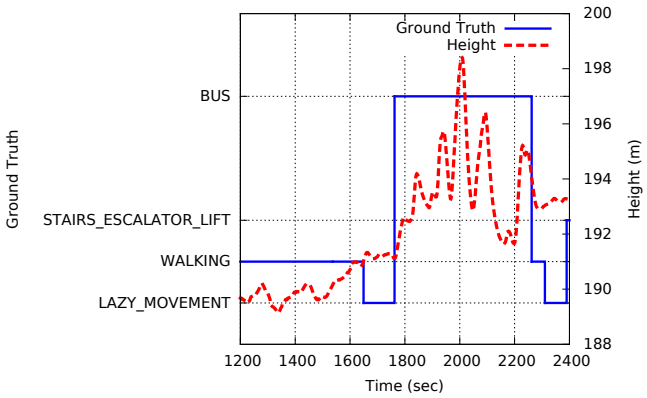
5.2 Intuition behind barometer context-detection

This section describes the intuitive logic based on which barometer context-detection is possible.

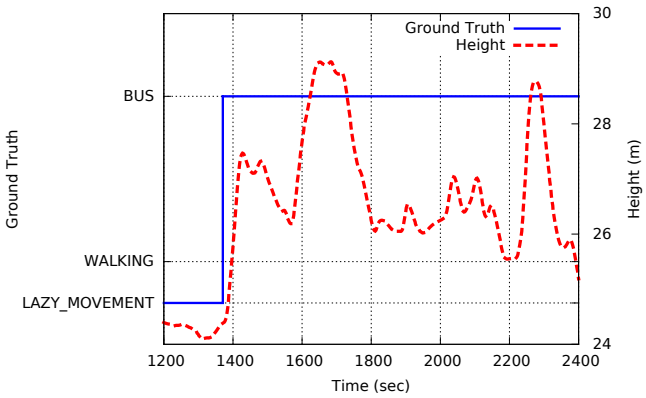
Roads are not perfectly flat. Their height changes slightly even when it is not visually obvious to the naked eye. The barometer is sensitive enough to measure this change in height when a vehicle moves along a road. Vehicle detection is based on the intuition that vehicles, because of their higher speed, tend to see more ups and downs and more rapid height changes, than walking in the same period of time (Note that vehicle bumps and jerks *do not* cause significant changes in height).

The graphs in Figure 2 explain this idea pictorially. The graphs are examples of traces we collected, that plot the variation in height value and user's ground truth versus time (In the graphs, the state *LAZY* is the same as *IDLE*). As can be observed, the number of ups and downs and the rate of height change increase considerably when the user is in a vehicle (Figures 2a, 2b and 2c). On the other hand, a user walking about in the same floor of a mall (considered as *IDLE* by our definition) sees very little change in height, and no ups and downs at all (Figure 2d). A user who is walking on a road does see change in height, but however does not experience the large number of ups and downs as in *VEHICLE*. The height variation between *WALKING*, *IDLE*, and *VEHICLE* can be best observed in Figure 2a.

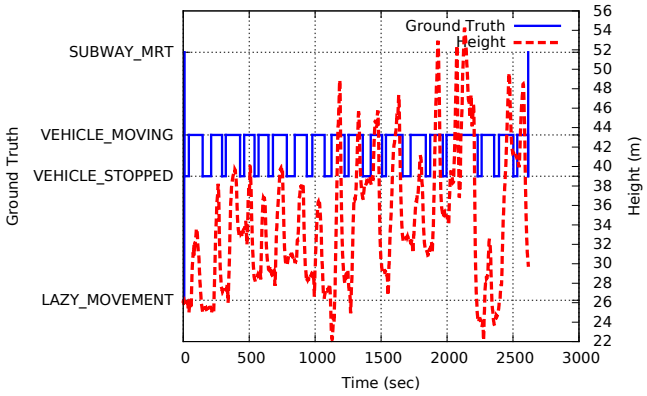
To summarise, barometer context-detection is based on the intuitive logic that users in vehicles see more rapid changes in height, including larger number of ups and



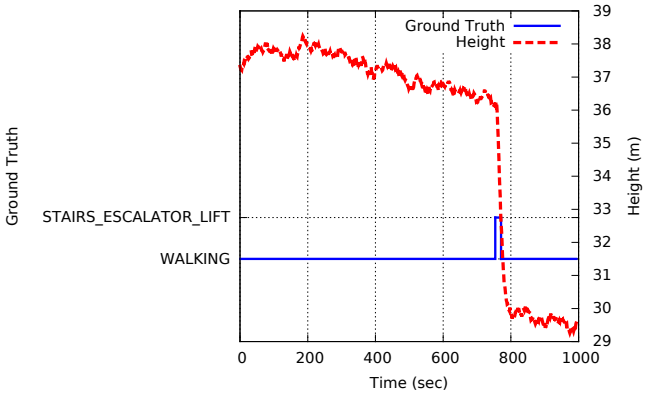
(a) Commute on a bus in Boston



(b) Commute on a bus in Singapore



(c) Subway ride in Singapore



(d) Walking between floors in a mall

Figure 2: Some examples of height variation against time for different user states

downs. Users who are walking see a gradual change in height. Users who are idle do not see any change in height.

5.3 Overview of Context-Detection Algorithm

Figure 3 gives a high-level overview of the barometer context-detection algorithm. It consists of four stages: Pre-processing, jump detection (Jumpdet), peak detection (Peakdet), and walk detection.

Jumpdet and peakdet are jointly responsible for *VEHICLE* detection. Based on the order of the stages, it can be seen that a determination of *VEHICLE* state overrides a determination of *WALKING* state. This is due to the fact that barometer is better at detecting vehicles than it is at *WALKING* (section 6.1).

The following sections discuss each stage of the algorithm in order.

5.4 Pre-processing

The barometer is sampled at a frequency of 1 Hz. Phones like Nexus 4 return values at a higher sampling rate (typically 4 Hz), in which case the sampling rate is clamped to 1 Hz in code. In phones like Nexus 5, due to internal smoothing, values are returned at a lower rate, usually every 2 seconds. In this case, linear interpolation is used to convert the actual rate into 1 Hz.

The pressure values returned by the barometer driver on Android is in millibar. This is converted to height in metres using the standard pressure-height formula [29]:

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

where h is the altitude in metres, while p and p_0 are the measured air pressure and sea level reference pressure respectively in millibar. This conversion of pressure into height is available as a helper method in Android. The sea level reference does not matter in our algorithm, since we only use relative height change.

The height values are typically noisy (1 metre noise on average). They are smoothened using a simple filter:

$$currentHeight = \alpha * sensorHeight + (1 - \alpha) * prevHeight$$

where α is 0.1. On Nexus 5, the barometer chip performs smoothing internally, and is not required in code. Figure 4 shows an example of the barometer sensor data after it has been smoothened.

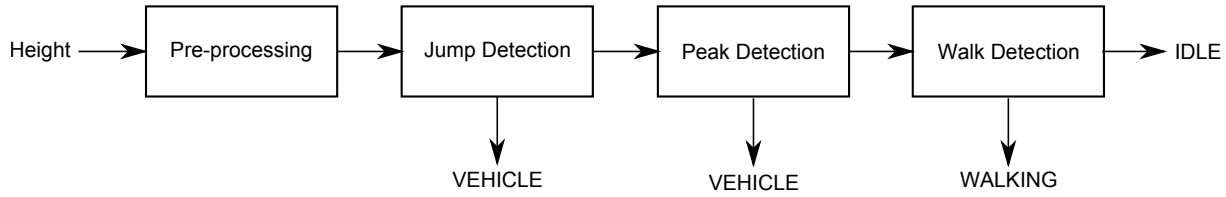


Figure 3: Overview of Barometer-based transportation context detection

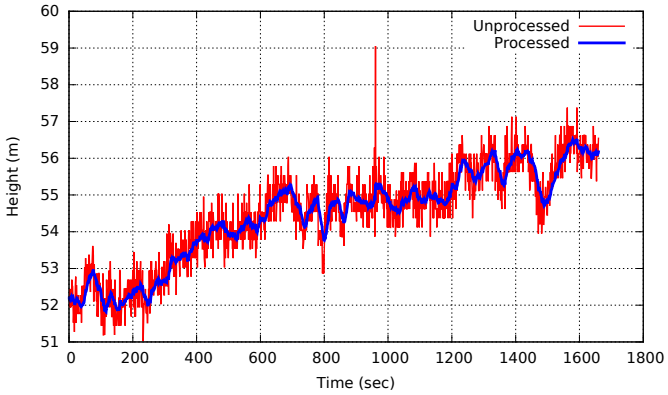


Figure 4: Barometer data after processing

5.5 Vehicle detection

The following two sections describe the logic used to detect the *VEHICLE* state, namely Jumpdet and Peakdet.

5.5.1 Jump detection (Jumpdet)

Vehicles that move at a high speed, or vehicles that move on highly sloped roads tend to observe a large rate of height change. In other words, they undergo 'jumps' in height, both in the up and down direction. We can detect such vehicles using what we term as 'Jump Detection' (Jumpdet).

We define a 'jump' as a height change of more than 0.8 metres in 5 seconds. For every height reading obtained from the barometer (1 Hz), we check if there is jump in height, by calculating the difference with a height reading obtained 5 seconds earlier. We keep track of the number of jumps observed in a 200 second sliding window, along with their sign (up or down).

Our experiments have shown us that walking does not cause height jumps, even on sloped roads, since the speed of walking is limited. An exception to this is when a person takes the stairs/escalator or elevator, all of which make it easier for a person to gain/lose vertical height with less effort. We can distinguish this from vehicle using the intuition that vehicles move both up and down, unlike a person that moves *only* up or *only* down the stairs/elevator.

When we observe that the ratio of the number of positive to negative jumps in the sliding window is in the range [30%, 70%], we classify the movement as vehicle. For the ratio to be meaningful, we calculate it only when the total number of jumps is more than 10. By capping the ratio to [30%, 70%], we ensure that we are seeing both positive as well as negative jumps in the sliding window of time, before deciding that the user's state is vehicle.

5.5.2 Peak detection (Peakdet)

Vehicles may move at a slower speed on roads that are not highly sloped, due to slow traffic or frequent stops. In such cases, Jumpdet fails to work since the rate of height change is not large enough. Instead, we detect such vehicles using what we term as 'Peak detection' (Peakdet).

Peakdet is based on the intuition that vehicles, due to their higher speed, observe a larger number of ups and downs in a given period of time compared to a person on foot. Every height reading from the barometer is fed through peak detection, a simple but online signal processing algorithm that can detect peaks and valleys in a signal of specified vertical size (in our case, we set the size to 1 meter).

While roads can be hilly, they are not undulating, i.e. the distance between ups and downs in a road are large and not small. Our experiments have shown us that a person on foot takes more than 200 seconds to cover the distance between undulations on highly sloped road, due to the large distance between them and slow walking speed.

Peakdet keeps track of the number of peaks and valleys in the height signal in a 200 second sliding window. If the number of peaks is more than 1, then the movement is classified as vehicle.

5.6 Walk and Idle detection

A user who is walking would see gradual rather than rapid changes in height (exceptions are while travelling on the stairs/escalator/elevator). The detection of *IDLE* and *WALKING* is based on the fact that height varies while walking, but is stable while idle.

Our algorithm calculates the stddev value of height in a 200 second sliding window. When the stddev rises above a threshold (0.3 metres), we classify the movement as walking, and idle otherwise.

Due to weather drift, the height tends to change over time, even when the user is idle. However, weather drift occurs over a larger time scale, and does not affect our algorithm which runs on a time scale of 200 seconds. Section 6.1.2 analyses the affect of extreme weather on our algorithm.

5.7 High-level Stitching

The vehicle detection described in the previous sections leads to a fragmented output due to long vehicle stops (for example at traffic lights). These fragments need to be stitched together to capture the transportation context at the user's level. We employ a simple stitching algorithm: Two vehicle detections occurring less than 2 minutes apart are stitched together. This simple algorithm stitches the vehicle

states together into a transportation journey more meaningful to the user and applications. To avoid false vehicle detections while idle or walking, vehicle state detections of less than 30 seconds duration are ignored.

5.8 Choice of thresholds and window sizes

In our algorithm, we use four configuration values, namely: the jump threshold (0.8 metres), the peakdet vertical height threshold (1 metre), the walking stddev threshold (0.3 metres), and the sliding window size (200 seconds). These values are characteristic of the overall terrain of the land.

For our algorithm, we have chosen the configuration values experimentally based on 10 hours of barometer traces collected before our evaluation's traces. In our evaluation using real-world traces and map elevation data, we find that the same configuration values however work well even in places with different terrains (section 6), and find that the algorithm performs well even without re-calibration.

6 Evaluation

We evaluate our work based on accuracy, power consumption, and latency, comparing it with two other approaches:

1. **Google Activity Recognition:** This is an accelerometer-based context detection algorithm implemented by Google [6]. Released in May 2013, it is part of the Google Play API, and is capable of detecting the modes *IDLE*, *WALKING*, *VEHICLE*, and *CYCLING*. Since it runs at very low power, we use this as a baseline for power consumption. The activity detection runs at an update interval specified by the application developer. Unless otherwise specified, in our evaluation the update interval is set to 10 seconds, since that is the highest frequency possible. We have used the Activity Recognition algorithm present in Google Play Services version 4.3, which was the latest version available in March 2014 when the trace data was collected.
2. **Future Urban Mobility Survey (FMS) App:** This is a GPS and accelerometer-based context detection implemented by the Singapore-MIT Alliance for Research and Technology (SMART) group in Singapore, in conjunction with the Singapore government's Household Interview Travel Survey (HITS). They have developed a smartphone application for iOS and Android, called FMS, to conduct household travel surveys as an alternative to the traditional HITS approach of conducting in-person interviews. It is the first survey application to be deployed and tested on a large scale, on over 1000+ users in Boston and Singapore. Since volunteers took both the FMS and HITS survey, the accuracy of the app has been thoroughly validated using the ground truth. The FMS application works by uploading sensor data to the FMS server, which in turn runs the context-detection using machine learning. Users can access the FMS website to validate the travel information. To reduce power consumption and minimize data upload, the FMS application duty cycles even the accelerometer, and uses a very low sampling rate of 2 Hz when it is turned on. The accelerometer in turn is used as a trigger for the GPS, which is sampled at 1 Hz, and duty cycled even during the vehicle journey.

Our evaluation is based on 178 hours of barometer traces (of which 47 hours are transportation journey traces) collected from 3 countries with the help of 13 volunteers. 15 hours of data was collected from Singapore by 7 volunteers, while 10 hours of data was collected from Boston (USA) by 6 volunteers. In addition, 22 hours of data was collected from a cross-country train in China by a single individual. During data collection, barometer sensor data was logged to the `sdcard` of the phone at a frequency of 1 Hz. For Singapore, volunteers additionally collected context output from the Google algorithm and FMS application.

Table 4 provides a summary of the traces collected from each country. There are more than 32 hours of vehicle activity and 15 hours of walking activity in total. As volunteers in Boston often leave the apps running on the phone for a long period of time while they are in office after they have completed their journeys, there are more than 40 hours of idle or stationary activity in this set of traces. This was also the case in China, with 85 hours of idle activity.

Volunteers were instructed to run the apps while carrying phones during their daily commute. No special instructions were given on how to carry the phones, and none of the journeys were decided beforehand. Volunteers recorded the ground truth by pressing buttons on the phone. Ground truth included activity at the user's level; low-level ground truth such as vehicle stops and minor walking stops were not logged. Use of this high-level ground truth yields a more realistic analysis of performance of the activity detection algorithms at the user's level.

In addition to these journey traces, we have also collected additional barometer traces to analyse the effect of weather drift (section 6.1.2), and to compare the accuracy of barometer algorithm with Google in a special case of the *IDLE* state (waiting) in section 6.1.3, and in a special case of the *VEHICLE* state (subway) in section 6.1.4.

The phones used by volunteers include Nexus 5, Nexus 4, Galaxy S3, and Galaxy S4, all updated to Jelly Bean. Our barometer-based detection algorithm processes the collected barometer data via a discrete-event trace-based simulator. The simulator is written such that the barometer algorithm code written in the simulator can be directly copied into the phone's application code. Since the simulator is deterministic and operates at a relatively large time scale (1 second) compared to processing, the output of the simulator matches that of running on the phone. Several traces have been checked to see that this is indeed the case, to ensure simulator correctness.

The Google and FMS algorithms are not simulated. The output of Google's detection is logged into a file in the phone, while the FMS detection results are available on the FMS website in the 'Activity Diary' of each user.

6.1 Accuracy

Accuracy is calculated by splitting each trace's timeline into intervals, and checking whether the context determined

Table 4: Summary of collected sensor trace data

Country	Volunteers	Total hours	Vehicle hours	Walking hours	Idle hours
Singapore	7	15	6.5	6.4	2.1
Boston (USA)	6	55.95	3.75	7.8	44.4
China	1	108.5	22	1.5	85

Table 5: Barometer algo versus Google (Accl) and FMS (GPS+Accl) Detection in Singapore

	Baro	FMS	Google	GoogleSmooth
Idle	76%	33%	76%	76%
Walking	54%	46%	79%	91%
Vehicle	81%	90%	31%	34%
Overall	69%	68%	56%	62%

Table 6: Barometer algo versus Google (Accl) detection in China

	Baro	Google	GoogleSmooth
Idle	99%	97%	98%
Walking	23%	40%	50%
Vehicle	78%	24%	25%
Overall	93%	82%	83%

by the three algorithms matches the ground truth in each interval. The accuracy is calculated as the fraction of the total number of intervals that the user activity is correctly identified. An interval size of 200 seconds was chosen to effectively eliminate intermittent short-duration states (our accuracy calculation discards any interval where there is a transition in the ground truth).

For fairness of comparison, we have implemented a majority voting scheme for smoothing of Google's fragmented output using a window size of 200 sec (this gave better results compared to smaller window sizes). In this paper, we refer to the smoothened Google algorithm as 'GoogleSmooth'.

Since all three algorithms are only available for Singapore, we will first present the accuracy comparison for this set of traces. Table 5 summarizes the results of the accuracy calculation for the traces from Singapore, for each algorithm and user state. Clearly, each algorithm has its own strengths and weaknesses. Google activity recognition, which is accelerometer-based, performs well for *IDLE* (76%) and *WALKING* (79%) detection, but doesn't perform well for *VEHICLE* detection, achieving only 31% accuracy (the evaluation in Section 6.1.4 further illustrates this point). FMS, due to its use of GPS, has good *VEHICLE* accuracy (90%), but performs poorly in *IDLE* and *WALKING* detection. Both accuracies are below 50%. Our barometer-based approach does well for *VEHICLE* and *IDLE* detection, due to its independence of vehicular vibrations and user movements. The accuracy for *WALKING* state detection is however much lower.

Table 7 shows the confusion matrix for barometer detection for Singapore. There are two factors that contribute to lower *WALKING* accuracy. Due to the long sliding window used in barometer detection, the algorithm has a significant latency before it decides the user has gotten off the vehicle. *WALKING* states which usually happen right after *VEHICLE* states are classified incorrectly. The second factor is the terrain of the land. Sometimes, roads may not be sloped enough for the height to significantly change while walking. This is an inherent limitation of our approach. If the threshold for height change is set too low, then changes in barometer readings caused by drift or environmental changes will be wrongly classified as walking. We fix the WALKING detection problem by fusing barometer and accelerometer in Section 6.5.

Tables 8 and 9 show the confusion matrices for Google activity recognition and FMS. The main source of error in Google's approach is in the *VEHICLE* detection. As the vehicle journey includes rides on subway which can be smooth between stations, and periods of traffic congestion on bus where movement is slow, the algorithm maps these vehicles states to the idle state in 38% of the cases. As for the FMS approach, it is unable to correctly determine the difference between idle and walking states in many instances.

The reason for the low detection accuracy of *IDLE* and *WALKING* by the FMS application is two-fold. First, it samples the accelerometer at a frequency of 2 Hz, which is much lower than related work in the literature, in order to save power and amount of data uploaded. Second, to reduce CPU awake power consumption, it duty cycles even the accelerometer, and hence there are periods of time where accelerometer data may be unavailable. Note that unlike majority of related work, FMS is a real large-scale deployment, and needs to prioritize power over accuracy.

Table 6 compares the barometer approach with Google's algorithm in China. As in the case of Singapore, Google performs relatively better for *WALKING*, while our barometer algorithm has better accuracy for *VEHICLE*, in contrast to Google's poor *VEHICLE* accuracy. Majority of vehicle data from China was collected in a cross-country train, in order to cover larger area of terrain. These results convince us further that sufficient terrain variations indeed occur for barometer to detect user activity.

Note that the *WALKING* detection accuracy of Google's algorithm is much lower in China (40%) compared to Singapore (79%) since the volunteer in China was with his kids, and may not have marked the ground truth for the short *WALKING* periods as accurately as for the long *VEHICLE* cross-country train. For Singapore and Boston, on the other hand, any trace where the ground truth was discovered to be marked incorrect during interview with the volunteer was

Table 7: Confusion Matrix for Barometer Algo

	Idle	Walking	Vehicle
Idle	76%	19%	5%
Walking	19%	54%	27%
Vehicle	6%	13%	81%

Table 8: Confusion Matrix for Google

	Idle	Walking	Vehicle	Unknown
Idle	76%	0%	0%	24%
Walking	10%	79%	0%	11%
Vehicle	38%	6%	31%	25%

discarded. For China, however, we did not discard the trace since only a single trace was available.

The effect of smoothing on Google's context detection (GoogleSmooth) can be seen in Tables 5 and 6. While smoothing improves WALKING detection, the VEHICLE accuracy remains low since Google outputs IDLE majority of the time in subways/trains. Consequently, the overall accuracy did not improve much for Google by smoothing.

6.1.1 *Location dependence*

In this section, we evaluate the accuracy for Boston and China using the algorithm settings designed for Singapore. This allows us to test how sensitive the setting of our algorithm is, and also allows us to test the barometer value sensitivity at different locations. Note that besides having different terrain variation, Boston has a very different weather pattern, in particular temperature ranges, than Singapore (Boston traces were collected during the polar vortex in 2014). Table 10 shows the accuracy of barometer-based context-detection for all 3 countries together. Note that the *IDLE* traces of Boston and China are included in the accuracy calculation, to check for any effect of weather patterns on *IDLE* detection.

As we can see from the table, accuracy of *VEHICLE* and *IDLE* detection still remained high, indicating that barometer-based detection can potentially work well in different locations even without re-calibration. However, the *WALKING* detection remains low and indicates a need for addition sensor fusion technique to complement the barometer-based sensing to improve accuracy. This will be discussed in Section 6.5.

Table 9: Confusion Matrix for FMS

	Idle	Walking	Vehicle
Idle	33%	34%	33%
Walking	37%	46%	17%
Vehicle	6%	4%	90%

Table 10: Barometer Algo accuracy for different locations

	Singapore	Boston	China
Idle	76%	85%	99%
Walking	54%	40%	23%
Vehicle	81%	72%	78%
Overall	69%	79%	93%

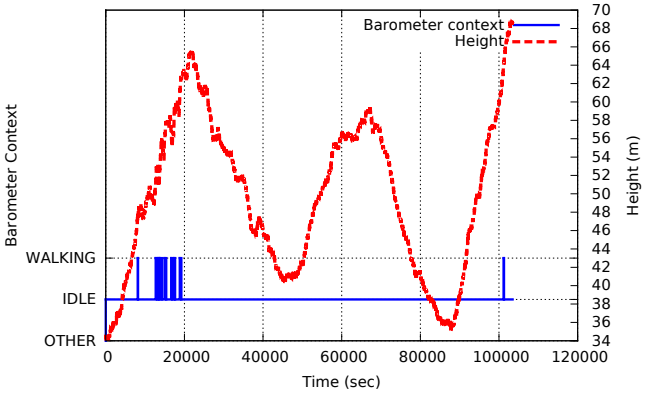


Figure 5: Barometer context detection during a rainy day (ground truth = IDLE). Shows diurnal pressure cycles.

6.1.2 *Weather dependence*

Unlike accelerometer and GPS, barometer is affected by drift due to weather. To analyse the effect of weather on barometer detection, we separately collected 29 hours of traces with raining and windy weather conditions from Singapore. The phone was kept idle and we check if changes in barometer readings under such weather conditions can trigger false positives and detect *IDLE* as either *WALKING* or *VEHICLE*.

The accuracy of *IDLE* detection was found to be 96%, indicating that weather does not have a significant impact on the barometer detection algorithm. This is due to the fact that weather drift tends to be on a larger time scale than the sliding window of our detection algorithm. Rather than cause unpredictable height variations as one might expect, weather drift is usually in one direction, and gradual, typically following the diurnal pressure cycles. This is shown in Figure 5. Similarly, windy weather does not produce significant drift in barometer. An example of how the readings change is shown Figure 6.

Finally, during the period of traces collection in Singapore and Boston, there are periods of dry weather (extended periods without rainfall), heavy thunderstorms, and snowfall. This further demonstrates that our approach of using barometer sensors to determine relative height change is not significantly affected by weather patterns.

6.1.3 *Accuracy of WAIT state*

To illustrate how barometer can be advantageous for *IDLE* detection, we collected a separate 30 minute trace of

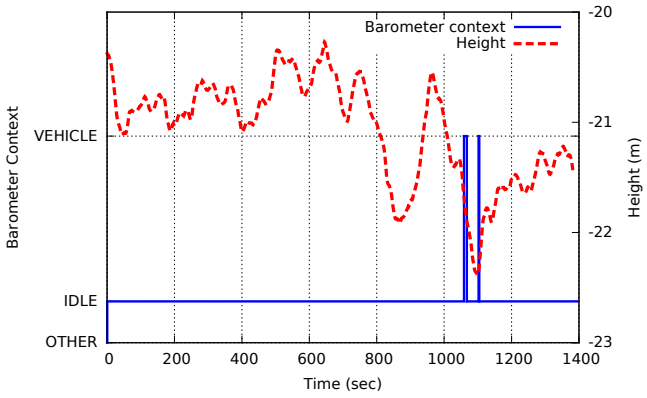


Figure 6: Barometer context detection during a windy day (ground truth = IDLE)

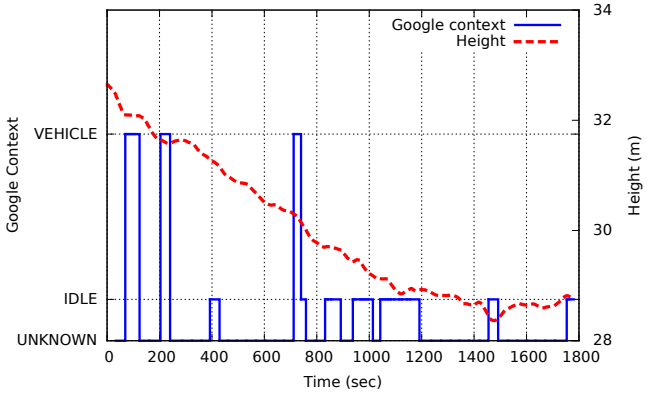


Figure 7: Google context detection while waiting at bus stop for 30 minutes (ground truth = IDLE)

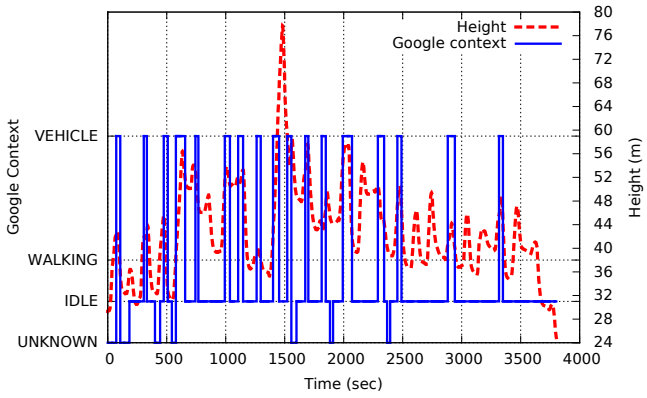


Figure 8: Google context detection on Subway (ground truth = VEHICLE)

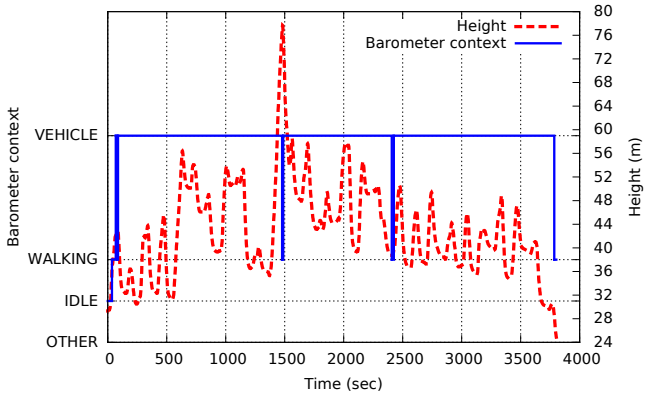


Figure 9: Barometer context detection on Subway (ground truth = VEHICLE)

waiting at a bus stop, running both Barometer and Google detection algorithms, while keeping the phone in hand.

We observed that even minor movements of the hand (while web browsing, for example), causes the accelerometer-based Google algorithm to get confused and output the state as *UNKNOWN* (see Figure 7). Consequently, it has a low accuracy for *IDLE* detection of just 25%. Barometer, in contrast, is unaffected by hand movements, and the accuracy is almost 100%.

6.1.4 Accuracy of subway state

Use of accelerometer to determine vehicle movement depends significantly on the movement patterns of the vehicle in use. This is unlike the use of barometer, where height change is relevant. To illustrate this issue, we collected a vehicle trace on the subway, running both barometer and Google algorithms.

The subway ride between two stations is generally quite smooth with limited vibrations and jerks, except near stations. The accelerometer-based Google algorithm has difficulty classifying the user's state as *VEHICLE*, and has a poor accuracy of 15%. A sample 1 hour subway ride segment is shown in Figure 8 where Google's approach often classifies the state as *IDLE*.

On the other hand, barometer relies only on air pressure to detect height change. This approach has a high detection accuracy of almost 100%. While it may not be obvious to a human, there is substantial height variations in the paths travelled by the subway both in underground tunnels and on above the ground train tracks, making it easy for the barometer to detect a subway ride. The same activity segment but with output states determined by our algorithm is shown in Figure 9.

The better accuracy of barometer over accelerometer in vehicle can be additionally observed from the results in the China cross-country train, where Google had a poor accuracy of 24%, while barometer accuracy was 78%. To summarize, although accelerometer works well in vehicles with substantial vibrations (such as in cars and buses), it performs poorly in smooth vehicles like subways. Barometer works well irrespective of the type of vehicle.

6.1.5 Note on Google's Algorithm

The version of Google's algorithm used in our evaluation was Google Play Services 4.3, the latest version available at the time of collection of trace data in March 2014.

Google appears to have significantly modified their activity recognition algorithm in subsequent releases of Google Play Services, which can now distinguish between *WALKING* and *RUNNING*. The algorithm is also more sensitive to phone movement, enabling it to better detect subway rides. We compared the accuracy of Version 4.3 (March 2014) and Version 5.0 (August 2014)[2] in the subway. The new version has a significantly higher accuracy of 81%, compared to the older version with accuracy of 22%.

However, the extra sensitivity of the new version also leads to a larger number of *VEHICLE* false positives caused by minor hand movement. We compared the accuracy of the old and new version while waiting for a bus. The old version detected 3% *VEHICLE* false positives, while the new version performed poorly with 88% *VEHICLE* false positives (Note that both the old and new version have a low *WAITING* detection accuracy, since the old version reports *UNKNOWN* most of the time, as observed in Section 6.1.3). This comparison shows that while the extra sensitivity in the new version enables better subway detection, it causes a large number of false positives due to minor hand movement when the user is idle.

6.2 Simulation using Map elevation data

In addition to real-world trace data, we have manually pulled over 900 km (30,000 data points) of elevation data from non-overlapping roads of 5 cities from Google Maps. Using map elevation data, we can evaluate the accuracy of our algorithm in a larger number of places over larger geographic areas having substantially different terrains, especially where we do not have real-world trace data available. An additional advantage is that we can vary the speed of travel and check the effect on accuracy.

Map elevation data was collected from roads in Kansas City (USA), San Francisco (USA), Lausanne (Switzerland), as well as from Singapore and Boston. Data was collected at 30 meter points, which is the highest resolution possible. Elevation data in between points are interpolated. This data has been used to emulate barometer sensor data, and fed to our algorithm to evaluate accuracy.

Table 11 shows the accuracy of barometer algorithm at different speeds. Two walking speeds (5 and 8 kmph) and three vehicle speeds (25, 35 and 50 kmph) are considered. 5 kmph is the average walking speed of a person, while 8 kmph is a fast pace. Note that the accuracy of vehicle for speeds higher than 50 kmph is expected to be higher, since number of ups and downs encountered would also be higher, and is hence not shown in Table 11.

Note that even in places like Kansas, sufficient terrain variations occur while travelling for barometer to distinguish between user states. Even at very low vehicle speed of 25 kmph, accuracy is high, which increases with increasing speed. Walking detection accuracy is lower, but can be fixed using fusion with accelerometer (Section 6.5).

[2]The apk of every Google Play Services release is available online and can be installed for accuracy comparison

Table 12: Comparison of terrain characteristics (stddev in brackets)

	Avg Elevation Change (m)	Avg Peak Distance (m)
Kansas City	0.84 (0.99)	479 (494)
San Francisco	1.05 (1.17)	645 (709)
Lausanne	1.04 (1.19)	395 (536)
Singapore	0.69 (0.65)	332 (252)
Boston	0.56 (0.66)	476 (435)

Table 13: Latency (sec) for each user state for barometer and Google algorithms (stddev in brackets)

	Baro	Google
Idle	176 (142)	78 (66)
Walking	158 (138)	26 (24)
Vehicle	211 (192)	122 (135)

Table 12 compares the terrain characteristics of the 5 cities, calculated using over 30,000 elevation data points, each spaced at 30 meters intervals, collected manually over non-overlapping roads from Google Maps. Average Peak Distance is the average distance between ups and downs on the road. In other words, if you plot a graph of elevation versus distance travelled, peak distance is the distance between two peaks in the graph. Smaller the value, more undulated the terrain, i.e. more peaks and valleys are encountered over the same distance travelled. From Table 12, Lausanne and Singapore have the most undulated terrain.

Average Elevation Change is the average change in elevation for every 30 metres of distance travelled. A higher value of average elevation change indicates higher road steepness. From Table 12, San Francisco and Lausanne have the steepest roads.

The accuracy of our barometer-based detection algorithm using map elevation data over these different terrains gives us confidence that this approach can indeed be generalized to other cities as well.

6.3 Latency

Barometer and Google detection algorithms run in real-time on the phone. In this section we compare the latency of these algorithms (FMS, in contrast, uploads data to the server and does post-processing). Latency is calculated as the average delay between transition to a user state, and detection of that state by the algorithm in question (for example, the average delay between a person starting to walk and the walk activity being detected). Table 13 lists the latency for each user state for both Google and barometer algorithms.

Google algorithm's IDLE latency is low since it is has been designed to detect even short vehicle/walking stops. However, applications typically prefer to ignore short user stops (detected by Google's algorithm), and rather focus on longer stops (detected by our barometer-based al-

Table 11: Accuracy of barometer-based context detection algorithm using map elevation data at different speeds

	Vehicle (50 kmph)	Vehicle (35 kmph)	Vehicle (25 kmph)	Walk (5 kmph)	Walk (8 kmph)
Kansas City	96%	93%	89%	73%	56%
San Francisco	92%	90%	76%	74%	66%
Lausanne	84%	83%	79%	58%	50%
Singapore	99%	99%	98%	63%	32%
Boston	99%	97%	91%	66%	58%

gorithm) which indicate higher-level user activities like home/office/shopping.

For VEHICLE state, both Google and barometer algorithms have higher latencies (2 to 3.5 min). This is due to the poor VEHICLE detection of Google's algorithm and the long sliding window of our barometer-based algorithm. The impact of this latency on applications depends on the journey duration. Analysis of real-world bus trip data from Singapore over 2,256,911 bus trips shows that the average duration of a bus ride is 14 minutes, while the maximum duration can be as large as 156 minutes. For long bus trips, we expect the VEHICLE latency to be acceptable, since it is a fraction of the total bus trip duration.

Applications interested in WALKING state (Eg: Fitness apps) require low latency. We exploit the low latency of WALKING detection of Google's algorithm by fusing both barometer and Google algorithms together in Section 6.5.

Note that although Google's activity detection has overall lower latency than our proposed barometer-based detection algorithm, the output of Google's algorithm is highly fragmented, a consequence of it being too reactive to state change.

6.4 Power Usage

In this section, we compare power consumption of barometer context-detection to Google's Activity Recognition. Measurements were performed on Galaxy S3 using the Monsoon Power Monitor. Our application is run in the background after acquiring a wake lock to keep CPU processing on. The screen and all wireless interfaces as well as data sync are kept switched off. The Android OS is unmodified and unrooted version 4.3, which came bundled with the phone from the manufacturer.

Barometer data is sampled using the Android sensor manager API. Note that although we specify 1 Hz to the sensor manager, the Galaxy S3 driver returns data at a higher rate of 5 Hz. However, we clamp and process data at 1 Hz in code.

The Google algorithm does not run continuously. It runs for 5 seconds each time it is triggered. For an update interval of 10 seconds, the program is triggered every 10 seconds, runs for 5 seconds, and sleeps for the remaining 5 seconds (Figure 10b). In contrast, the barometer based context-detection runs continuously. The power consumption can be further reduced if sensor batching is utilized. This is dicussed in Section 7.

Since the accelerometer sampling rate and calculations involved are higher, the power usage is also correspondingly high. Barometer, on the other hand, uses lower power due to its low sampling rate and simple calculations involved (refer to Figures 10a and 10b).

Table 14: Power usage

	Power (mW)
CPU Idle	25
CPU Awake	85
Google	120
Baro	88

Table 14 shows the power consumption. The power values listed for barometer and Google approaches include the base CPU awake power, sensing power, as well as computation power. Note that the Galaxy S3 phone used in Table 2 was a different phone, which is why the base CPU awake power is different, perhaps due to a difference in the Android version.

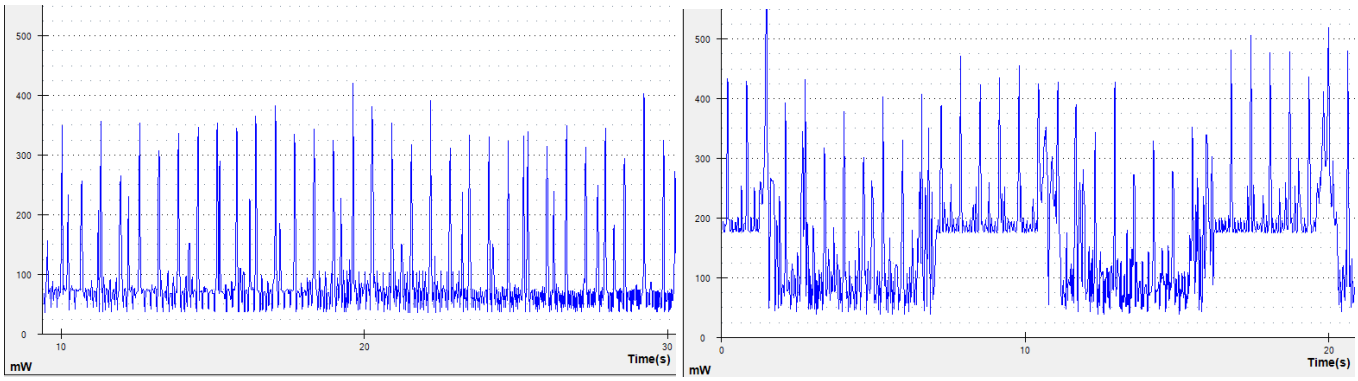
In spite of running continuously, the barometer-based algorithm consumes 32 mW lower power than the accelerometer-based algorithm. Google consumes 35 mW over the base power, while barometer uses only 3 mW over the base power, a significant improvement.

We had a detailed discussion with the authors of [25] on the measurement methodology. The power measurement methodology in their work differs significantly from ours in several aspects. First, they run the power measuring app in foreground rather than background. Second, they do not process sensor data but log sensor readings to the `sdcard`. Finally, the accelerometer and barometer were read and processed at different frequencies. Conversely, as we elaborated at the beginning of this section, we run our power measuring app in the background, process the sensor readings (either with Google's algorithm for the accelerometer or ours for the barometer), and do not save the readings to `sdcard`.

6.5 Fusion of barometer and accelerometer

We can fuse both barometer and accelerometer algorithms together to increase detection accuracy. Although the power consumption increases compared to using a single algorithm, with the advent of sensor hubs and offloading of activity detection into hardware, power consumption may reduce drastically, making it worthwhile to fuse multiple sensors for higher context detection accuracy.

In this paper, we found that barometer and accelerometer have complementary strengths and weaknesses: barometer is good for IDLE and VEHICLE detection, but poor in WALKING detection, while accelerometer is good for WALKING detection, but poor in IDLE and certain VEHICLE detections. A simple fusion technique can combine the strengths



(a) Power profile of barometer-based context detection

(b) Power profile of Google's context detection

Figure 10: Power profile of Google and barometer algorithms

Table 15: Fusing barometer and Google algorithms

	Baro	Google	Fusion
Idle	76%	76%	76%
Walking	54%	79%	88%
Vehicle	81%	31%	77%
Overall	69%	56%	81%

of both sensors, by first giving precedence to accelerometer for WALKING, and then to barometer for other states.

Table 15 shows the accuracy using fusion of the barometer and Google algorithms. The overall accuracy improves drastically over using a single sensor. Fusion also fixes the WALKING detection problem faced by barometer.

7 Discussion and Future Work

In this section, we discuss two issues pertaining to the use of barometer for context-detection:

7.1 Sensor Batching

The main source of power consumption in current activity detection algorithms is the need to keep a wake lock on the main processor to process sensor data continuously. Hardware implementations such as the M7 co-processor reduce this power, but are inflexible. A new hardware feature has been introduced in Nexus 5, called sensor batching, where sensor data can be buffered while the processor sleeps, and processed in a batch later when the processor wakes. This provides a nice balance between power and software flexibility. The number of readings that can be buffered is limited by the sampling rate and the size of the data. The low sampling rate of barometer (1 Hz) makes it excellent for sensor-batching, compared to 3-axial accelerometer which requires higher sampling rates and larger data. Barometer data can be buffered for several minutes even when using small-size buffers. With sensor batching, the barometer would become truly ultra-low power. Unfortunately, we were unable to tap into the battery of Nexus 5 for power measurements, and so are unable to provide an evaluation of the effect of sensor batching on the power consumption of barometer detection.

7.2 Combining Temperature with Pressure

One potential way to improve *WALKING* detection is the use of temperature along with pressure. Barometer chips contain embedded temperature sensors, so reading the temperature comes at no extra cost. Air pressure and temperature are co-related. When the phone is idle, a change in pressure would be associated with a corresponding change in temperature. While walking, this co-relation is disturbed since the change in pressure gets additionally affected by the change in altitude. This could act as an indicator for the *WALKING* state. In our measurements of pressure and temperature indoors on the Galaxy S4 (which has an ambient temperature sensor), this co-relation appeared to be correct. However, when we performed measurements outdoors while idle, this co-relation does not seem to hold, as seen in Figure 11. Ideally, even when outdoors, when the user is idle, pressure and temperature would show a co-relation. One reason this is not the case is perhaps the slower reaction of the temperature sensor to change in temperature, or low resolution. Further exploration of this idea using higher quality and higher resolution sensors is left as future work.

7.3 Integration with FMS App

We are looking into integrating our algorithm into the FMS App, to eliminate the 'movement' false positives often generated by accelerometer which trigger the high-power location service. This can help reduce power consumption since the location service will not run unnecessarily.

8 Conclusions

In our work, we demonstrate an alternative approach for low-power transportation context detection using only the barometer sensor, a relatively new sensor now present in an increasing number of phones. Unlike accelerometer, the predominant sensor in use today, barometer is unaffected by phone position and orientation, but instead depends only on the overall terrain of the land. Using a low sampling rate of 1 Hz, and simple processing based on intuitive logic, we show that barometer can be used for the detection of the states *IDLE*, *WALKING*, and *VEHICLE* while consuming 32 mW lower power than even the accelerometer, at the same time

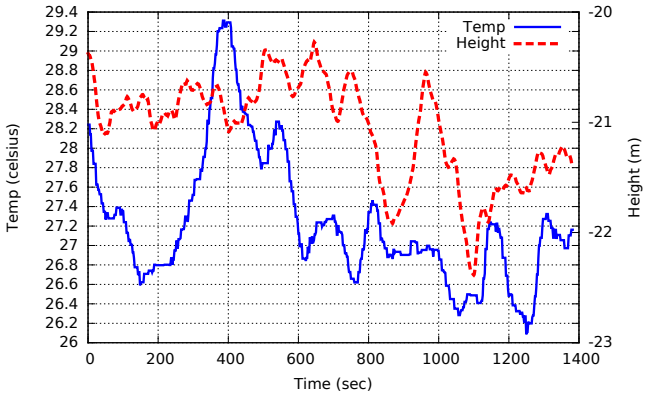


Figure 11: Variation of temperature and pressure outdoors when IDLE

achieving comparable accuracy to Google's Activity Recognition algorithm and the FMS application. Barometer also solves the problems of accelerometer in detecting the *WAITING* state, and certain vehicles like the subway. Finally, we found that fusion of barometer and accelerometer gives the best accuracy by combining the strengths of each sensor.

9 Acknowledgments

We would like to thank the anonymous reviewers and our shepherd, Lama Nachman, for their valuable comments and suggestions. We would also like to thank Fang Zhao and Ajinkya Ghorpade of the FMS team for helping us in the evaluation, as well as all the volunteers who helped us collect sensor trace data. Special thanks to Jason Gao from MIT for his help. This research was supported in part by the National Research Foundation Singapore through the Singapore-MIT Alliance for Research and Technology (SMART) program.

10 References

[1] Cover is an android-only lockscreen that shows apps when you need them. http://techcrunch.com/2013/10/24/cover-android/.

[2] Google's Fused Location API, Google I/O 2013. https://www.youtube.com/watch?v=URcVZybzMUI.

[3] Apple M7. http://en.wikipedia.org/wiki/Apple_M7.

[4] Google adds low-power step counting to android 4.4. http://mobihealthnews.com/26977/google-adds-low-power-step-counting-to-android-4-4/.

[5] Moves application. http://www.moves-app.com/.

[6] Google's Activity Recognition API. http://developer.android.com/google/play-services/location.html.

[7] OzlemDurmaz Incel, Mustafa Kose, and Cem Ersoy. A Review and Taxonomy of Activity Recognition on Mobile Phones. *BioNanoScience*, 3(2):145–171, 2013.

[8] Weathersignal. https://play.google.com/store/apps/details?id=com.opensignal.weathersignal.

[9] Pressurenet. https://play.google.com/store/apps/details?id=ca.cumulonimbus.barometernetwork.

[10] Carlos Carrion, Francisco Pereira, Rudi Ball, Fang Zhao, Youngsung Kim, Kalan Nawarathne, Naijia Zheng, Chris Zegras, and Moshe Ben-Akiva. Evaluating fms: A preliminary comparison with a traditional travel survey. In *93rd Annual Meeting of the Transportation Research Board*, 2014.

[11] Caitlin D Cottrill, Francisco Camara Pereira, Fang Zhao, Ines Ferreira Dias, Hock Beng Lim, Moshe E Ben-Akiva, and P Christopher Zegras. Future mobility survey: Experience in developing a smartphone-based travel survey in singapore. *Journal of the Transportation Research Board*, 2354:59–67, 2013.

[12] Sasank Reddy, Min Mun, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. Using mobile phones to determine transportation modes. *ACM Trans. Sen. Netw.*, 6(2):13:1–13:27, March 2010.

[13] Jason Ryder, Brent Longstaff, Sasank Reddy, and Deborah Estrin. Ambulation: A tool for monitoring mobility patterns over time using mobile phones. In *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 04*, CSE '09, pages 927–931, Washington, DC, USA, 2009. IEEE Computer Society.

[14] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. Understanding mobility based on gps data. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, UbiComp '08, pages 312–321, New York, NY, USA, 2008. ACM.

[15] Ian Anderson and Henk Muller. Practical activity recognition using gsm data. Technical Report CSTR-06-016, Department of Computer Science, University of Bristol, July 2006.

[16] Timothy Sohn, Alex Varshavsky, Anthony LaMarca, MikeY. Chen, Tanzeem Choudhury, Ian Smith, Sunny Consolvo, Jeffrey Hightower, WilliamG. Griswold, and Eyal Lara. Mobility detection using everyday gsm traces. In Paul Dourish and Adrian Friday, editors, *UbiComp 2006: Ubiquitous Computing*, volume 4206 of *Lecture Notes in Computer Science*, pages 212–224. Springer Berlin Heidelberg, 2006.

[17] Sarfraz Nawaz, Christos Efstratiou, and Cecilia Mascolo. Parksense: A smartphone based sensing system for on-street parking. In *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, MobiCom '13, pages 75–86, New York, NY, USA, 2013. ACM.

[18] Jeffrey Hightower, Sunny Consolvo, Anthony LaMarca, Ian Smith, and Jeff Hughes. Learning and recognizing the places we go. In *Proceedings of the 7th International Conference on Ubiquitous Computing*, UbiComp'05, pages 159–176, Berlin, Heidelberg, 2005. Springer-Verlag.

[19] Shuangquan Wang, Canfeng Chen, and Jian Ma. Accelerometer based transportation mode recognition on mobile phones. In *Proceedings of the 2010 Asia-Pacific Conference on Wearable Computing Systems*, APWCS '10, pages 44–46, Washington, DC, USA, 2010. IEEE Computer Society.

[20] Pekka Siirtola and Juha Röning. Recognizing human activities user-independently on smartphones based on accelerometer data. *International Journal of Interactive Multimedia & Artificial Intelligence*, 1(5), 2012.

[21] Samuli Hemminki, Petteri Nurmi, and Sasu Tarkoma. Accelerometer-based transportation mode detection on smartphones. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, page 13. ACM, 2013.

[22] Jieying Zhang, E. Edwan, Junchuan Zhou, Wennan Chai, and O. Loffeld. Performance investigation of barometer aided gps/mems-imu integration. In *Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION*, pages 598–604, April 2012.

[23] M. Tanigawa, H. Luinge, L. Schipper, and P. Slycke. Drift-free dynamic height sensor using mems imu aided by mems pressure sensor. In *Positioning, Navigation and Communication, 2008. WPNC 2008. 5th Workshop on*, pages 191–196, March 2008.

[24] S. Vanini and S. Giordano. Adaptive context-agnostic floor transition detection on smart mobile devices. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*, pages 2–7, March 2013.

[25] Kartik Muralidharan, Azeem Javed Khan, Archan Misra, Rajesh Krishna Balan, and Sharad Agarwal. Barometric phone sensors–more hype than hope! *15th International Workshop on Mobile Computing Systems and Applications*, 2014.

[26] Semefab Limited. MEMS Pressure Sensors: Technologies and Fabrication. 2011 Whitepaper.

[27] Stephen Ming-Chang Hou. *Design and fabrication of a MEMS-array pressure sensor system for passive underwater navigation inspired by the lateral line*. PhD thesis, Massachusetts Institute of Technology, 2012.

[28] *Hardware and software guidelines for use of the LPS331AP*, November 2012.

[29] Greg Milette and Adam Stroud. *Professional Android Sensor Programming*. Wiley, 2012.