# Wolverine : Traffic and Road Condition Estimation using Smartphone Sensors

**Stage I Report**

Submitted in partial fulfillment of the requirements for

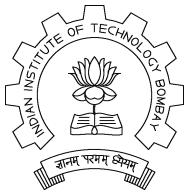
**B.Tech Project**

by

**Ravi Bhoraskar**
**Roll No: 08005002**

under the guidance of

**Prof. Bhaskaran Raman**




Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai

## Abstract

*Monitoring road and traffic conditions in a city is a problem widely studied. Several methods have been proposed towards addressing this problem. Several proposed techniques require dedicated hardware such as GPS devices and accelerometers in vehicles [8][17][10] or cameras on roadside and near traffic signals[15]. All such methods are expensive in terms of monetary cost and human effort required. We propose Wolverine[1] - a non-intrusive method that uses sensors present on smartphones. We extend a prior study [14] to improve the algorithm based on using accelerometer, GPS and magnetometer sensor readings for traffic and road conditions detection. We are specifically interested in identifying braking events - frequent braking indicates congested traffic conditions - and bumps on the roads to characterize the type of road. We evaluate the effectiveness of the proposed method based on experiments conducted on the roads in Mumbai, with promising results.*

[1] An allusion to the comic book character Wolverine from X-Men, who has heightened sensing powers

# Contents

# Chapter 1

# Introduction

With growing number of vehicle users, traffic is growing day by day. It is desirable to have a mechanism by which people can know, in real time, about the traffic condition in the routes on which they wish to travel. As a result, working on traffic monitoring has gained significant attention in recent times. Much of the previous work concentrated on lane system and orderly traffic [10], which is rare outside the developed world. For example, in India, the traffic is highly chaotic and unpredictable. Further, many of the proposed solutions need installing dedicated sensors in the vehicles (like GPS-based tracking units)[17], and/or on the road side (like inductive loop vehicle detectors, traffic cameras, Doppler radar, etc.)[15][16], which are expensive. Also, installing sensors in a large number of vehicles or installing traffic cameras at several junctions is impractical due to monetary cost and human effort required. The methods which use inductive loop vehicle detectors can only be used in laned traffic systems, which is not the case in many countries. In addition, traffic detection using traffic cameras is restricted to the location where they are deployed (generally at traffic signals). To detect the road conditions, accelerometers installed in a vehicle may be used[8]. But, this is also not feasible, as the number of vehicles participating in the system will be limited to the vehicles where accelerometers is installed. Installing additional devices on vehicles to increase sensing density can very quickly become prohibitively expensive.

Several methods have been proposed that use sensors in smartphones for activity detection in various environments(Indoor localization[12], traffic detection[14] and detecting activity of a person[13]). The smartphone based traffic estimation methods obviate the need for specialized hardware installed in vehicles or on the road side. These *crowdsourced* solutions(using distributed participatory data collection) have the advantage of high scalability as the number of smartphone users is growing at a rapid pace. The Nericell system [14] uses accelerometer, microphone, GSM Radio and GPS sensors available in smartphones that users carries with them. In a smartphone based method, the orientation of the phone could be arbitrary with respect to the direction of motion, and could also change repeatedly. Hence, it is required to virtually reorient the axes of the phone with respect to the vehicle. Nericell uses accelerometer and GPS readings alone for this. The direction of gravity is used to sense the vertical orientation, and the acceleration recorded during a braking event is used to compute the horizontal orientation. Autowitness[11], a system to track stolen property also uses an idea similar to Nericell in order to reorient the axes. Further, Nericell detects road and traffic conditions based on threshold based heuristics.

Wolverine is a method which is similar to the Nericell system in that it too uses smartphone sensors for traffic state monitoring. However, for axes reorientation, we use the magnetometer to find the horizontal orientation of the phone instead of waiting for a braking event. This makes the system more reliable, and also reduces the energy intensive GPS usage. We give an energy consumption model for Wolverine, and compare it to Nericell, showing significant decrease in energy consumption, thus prologing the battery life. Also, instead of threshold based heuristics for determining the traffic and road conditions, we use machine learning techniques (K-means clustering and Support Vector Machine (SVM)) which are more robust and versatile as compared to threshold based methods.

Hence, the main contributions of this work are two fold (1) A novel algorithm to virtually reorient the coordinate axes of a disoriented phone (2) Machine learning techniques to identify bump and braking events. (3) An energy consumption model comparing this method with Nericell.

This project is joint work with Nagamanoj Vankadhara. The division of work was as follows

- Reorientation Algorithm : *Nagamanoj*
- Machine learning based classification : *Ravi*
- Energy Consumption Model : *Ravi*
- Experimental Evaluation : *Ravi and Nagamanoj*

## Chapter 2

# Experimental Setup

Before we move on to describe our algorithms in detail, in this chapter, we describe the experimental setup that we used to collect the data, and to validate our algorithms.

Smartphones come with wide range of capabilities. They can process information at high speeds and can support good amount of external storage. The storage capability can be used for piggybacking processed or raw data collected by the application to estimate traffic conditions. The processed or unprocessed data can be communicated to a central server using GPRS, EDGE, UMTS, WiFi or Bluetooth. Many sensors - accelerometer, magnetometer, gyroscope, microphone, camera and GPS - can be used to sense the environment and location information.

We chose Android platform as it is open source and has a fairly versatile API (Application Programmer Interface). We used *Google Nexus S* and *HTC Wildfire S* both of which runs Android 2.3.3 OS and used SDK version 10. Both these phones are equipped with an accelerometer, a magnetometer and GPS sensor. *Nexus S* has a gyroscope as well.

We developed an application which processes the accelerometer and magnetometer readings along with GPS information and outputs the linear acceleration of the mobile, in the vehicle frame of reference. We used a Suzuki Access 125 vehicle, and collected data in different locations on the IIT Bombay campus. We chose different road conditions (smooth, bumpy, inclined) and environment conditions(clear sky, covered with trees - these can affect GPS readings).

# Chapter 3

# Virtual Reorientation Mechanism

When the phone is kept in proper orientation so that, its axes align with vehicle's axes, the readings of the three-axes accelerometer gives vehicle acceleration in all three directions. The phone can be kept in any arbitrary orientation inside the vehicle. For example, it can be kept in a phone holder, which is not flat, but in some orientation. We need to virtually reorient the axes of the phone so that they are aligned with the vehicle's axes. Once we reorient the axes of the phone, we use the acceleration readings to estimate traffic and road conditions.

## 3.1 Framework

The Android system assumes axes of the accelerometer as shown in Figure-3.1. The X-axis is along the smaller edge when the phone is held in portrait mode and points towards right, Y-axis is along the longer edge in portrait mode and points up and the Z-axis is perpendicular to the plane of the front face of the screen and points towards the sky when laid on table on its back. It also assumes same axes for magnetometer, except, the Z-axis is inverted and points vertically downwards towards center of the earth. We assume the vehicle's axes as shown in Figure-3.2. The direction of motion of vehicle is the Y-axis, the direction towards the right hand side of the driver will be X-axis and the direction which acts vertically upwards from the center of gravity is the Z-axis. We call the axes of the phone, the reference axes (X,Y,Z) and the axes of the vehicle, the target axes (X', Y', Z').

Both the accelerometer and magnetometer gives readings along all the three axes of the phone. Imagining the accelerometer as a small ball at the center of the phone, its readings represent the force applied by the phone against the force acting on the ball to displace it (values are in m/s$^2$). The magnetometer measures ambient magnetic field in the three axes (values are in micro tesla).

## 3.2 Determining Accelerometer Orientation

As described in Chapter 3, the accelerometer (or rather the mobile that contains this accelerometer) can be in any orientation in the vehicle. We need to find the readings of the accelerometer along Y' and Z' axes of the vehicle to find braking and bump events respectively, which we use for estimating traffic and road conditions. When the phone is kept idle in some arbitrary orientation, the accelerometer shows some non-zero readings, giving illusion that it the phone is accelerating in those directions with the specified acceleration, which is actually not accelerating. This is because the force of gravity which acts vertically downwards, is factorized along the axes of the accelerometer. Similarly, even when the device is actually accelerating some direction, the readings does not corresponding to only the actual acceleration, but also includes this added factor of gravity. We need to calculate this force acting along the axes of the phone in that orientation and subtract it from readings. This can be done once we know the orientation of the phone.

To transform the vector from reference coordinate system to target coordinate system, one method is to find the angles by which the axes of the reference coordinate system need to be rotated around each of the axes, to align with the axes of the target coordinate system. Each of these rotations can be expressed in the form of a rotation matrix. For example, consider the reference coordinate system XY in 2-D plane, is rotated at an angle $\alpha$ in counter-clockwise direction, then the rotation matrix equivalent to this rotation can be represented as

$$\text{R} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{3.1}$$

and, a column vector V in reference coordinate system can be transformed to new coordinate system just by multiplying it with the rotation matrix with it, as shown

$$V' = R(\theta) \times V \tag{3.2}$$

Similarly the rotation matrix corresponding to the rotations of the reference coordinate system at an angle $\phi$ around Y-axis, then at an angle $\psi$ around X-axis and then at an angle $\theta$ around Z-axis, can be formed by multiplying the rotation matrix

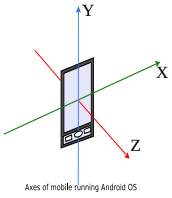



**Figure 3.1. Phone Axes**

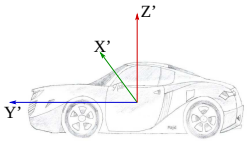



**Figure 3.2. Vehicle Axes**

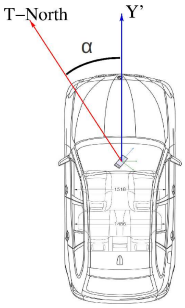



**Figure 3.3. Direction of Motion**

corresponding each rotation. A vector V in reference coordinate system can be represented in target coordinate system as

$$V' = R(\theta)R(\psi)R(\phi) \times V \tag{3.3}$$

$$\begin{bmatrix} v'_x \\ v'_y \\ v'_z \end{bmatrix} = \begin{bmatrix} c(\theta)c(\phi) + s(\theta)s(\psi)s(\phi) & -s(\theta)c(\phi) + c(\theta)s(\psi)s(\phi) & c(\psi)s(\phi) \\ s(\theta)c(\psi) & c(\theta)c(\psi) & -s(\psi) \\ -c(\theta)s(\phi) + s(\theta)s(\psi)c(\phi) & s(\theta)s(\phi) + c(\theta)s(\psi)c(\phi) & c(\psi)c(\phi) \end{bmatrix} \times \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{3.4}$$

where *c* represents *cos* and *s* represents *sin*.

Here we can observe that, each row vector in the rotation matrix is mutually perpendicular, and have unit magnitude. In fact, these are the axes representing the axes of the target coordinate system. Also, the magnitude of each column is unit, which means, each vector in old coordinate system will be split onto each of the axes of new coordinate system so that the total magnitude will not be changed. Thus, the rotation matrix can be formed by just knowing the axes of the target coordinate system (vectors representing the target coordinate axes with respect to reference coordinate system).

We use the second method to perform virtual reorientation of the axes of the accelerometer (transforming the accelerometer readings from phone's coordinate system to vehicle's coordinate system). We consider the phone's axes as the reference coordinate axes and the vehicle's axes as the target axes. We do the total reorientation in two steps. First, we transform the accelerometer readings from phone's coordinate system to geometric coordinate system, then from geometric coordinate system to vehicle's coordinate system.

## 3.3 Reorientation from Phone's axes to Geometric axes

The accelerometer can give *Gravity Vector*, which acts vertically downwards towards the center of the earth. The magnetometer can give the *Magnetic Vector*, which acts towards the magnetic north. Cross product of these two vectors gives us a vector perpendicular to the plane of these two vectors and acts along magnetic west (using right hand rule [4]). We call it *East West Vector*. Again cross multiplying the *Gravity Vector* with *East West Vector*, we get *North South Vector*, which acts horizontally towards magnetic north. We do not present the exact calculation here due to lack of space. Interested user may refer [5]. Now, we have three mutually perpendicular vectors, representing the geometric coordinate system. The rotation matrix representing this system as target system can be formed as,

$$\mathrm{R}^{-1} = \begin{bmatrix} \text{East West Vector} \\ \text{North South Vector} \\ \text{Gravity Vector} \end{bmatrix} \tag{3.5}$$

This rotation matrix is in fact a $3 \times 3$ matrix, which is equivalent to (1). Each of the row represent the projections of the geometric axes onto the phone's coordinate axes. The obtained rotation matrix represents the angles of rotation of the phone around the device's axes to align with geometric axes. The orientation of the phone with respect to geometric coordinate system can be calculated from the rotation matrix with simple trigonometric calculations. Once we have the three angles, we calculate the factor of gravity that is added accelerometer readings along each axis and subtract it. Multiplying the obtained rotation matrix with the column vector representing the modified acceleration values gives us the acceleration values of the phone along each of the axes of geometric coordinate system. i.e, we can get the acceleration values along magnetic west direction, magnetic north direction and along the gravity vector direction. Here we need to observe that, magnetic north and the true north are deviated by a small angle, hence the magnetic west and true west also. We will calculate this deviation in next section.

## 3.4 Reorientation from Geometric axes to Vehicle's axes

Once we have the acceleration values along the magnetic north and magnetic south, transforming them onto vehicle's axes is simple if we know the angle at which the direction of motion of the vehicle deviates from the magnetic north. We calculate the magnetic declination, which is the deviation of magnetic north from true north, which we can calculate from the

magnetometer readings and GPS readings, as shown in [2]. We also calculate the bearing as described in [1], which is the angle that a line joining two latitude-longitude points makes with geometric north as shown in the Figure-3.3. We can extract the latitude/longitude information from the location fixes returned by GPS. A GPS fix is the location identified by the GPS receiver. To reduce the possibility of miscalculation of this bearing due to the lack of accuracy in GPS fixes, we take average of bearings calculated from first fix to next 30 fixes.

Once we know the bearing and the magnetic declination, we can transform the values from geometric axes to vehicle's axes with the help of a rotation matrix representing a rotation of the geometric axes at an angle $bearing - declination$ .

# Chapter 4

# Road and Vehicle State Detection

Previous sensing methods use thresholds on accelerometer data for bump detection and sensing braking of the vehicle [14]. However, according to experiments we conducted, the characteristics of the acclerometer data changes with enviroment configuration. In particular, the vehicle, the mobile device and the nature of the road affect the characteristics of the sensor data. Due to this variation in characteristic, the accuracy of the system with fixed thresholds would be lower when tested under different conditions. The method we propose is to use machine learning algorithms to classify the incoming stream of accelerometer data into classes based upon the features present in the data itself. This will make the classification more robust to changes in the environment. In this chapter, we describe two techniques. The first one is used for classifying data from bumpy roads as separate from smooth roads. The second one is for detecting braking.

## 4.1 Bump Detection

The problem that we address in this chapter is to distinguish a bumpy road from a smooth one. This can be used both to detect single speed-bumps and lengthier patches where the road is consistently bumpy.

First the virtual reorientation, described in Chapter 3, is applied on the acclerometer data in order to convert it from the phone frame of reference to the vehicle frame of reference. Then we divide the reoriented accelerometer data into windows of 1 second duration, and use each window as a single data point (see Figure 4.1). The sampling rate of the accelerometer we used was 50 times per second. We found that while one second is large enough to smoothen out the random jitters in the data, it is also small enough to capture the inherent characteristic features of a bumpy or smooth road. From each of the 1 second window, we extract the features to be used as input to the machine learning algorithm. We use six features - the mean and standard deviations in the three coordinate axes ($\mu_X, \mu_Y, \mu_Z, \sigma_X, \sigma_Y$ and $\sigma_Z$) over the window.

We devised a technique that is essentially a supervised learning technique. However, it uses an unsupervised learning algorithm to help generate the class labels for the training. First, we use a *K-means clustering algorithm* [3], with $K = 2$, to partition the set of data points into two classes. The class labels are arbitrary at this point of time. These classes are then manually labelled as either *Bumpy* or *Smooth*. Notice that the manual labelling is done only for the two classes, and not for individual data points. However, the user might choose to re-label some points, based on his observations in the data collectio phase. This makes the effort required in this step considerably smaller than if he had to label every data point individually. After this manual labeling is done, we have a set of labeled data points. These are then used to train a *Support Vector Machine classifier* [6]. This trained SVM, in turn, is used to classify the data points that are generated during the test phase, and hence to predict the vehicle state.

We observe from Figures 4.1 and 4.2 that the major difference in characteristic of the data for the bumpy and smooth

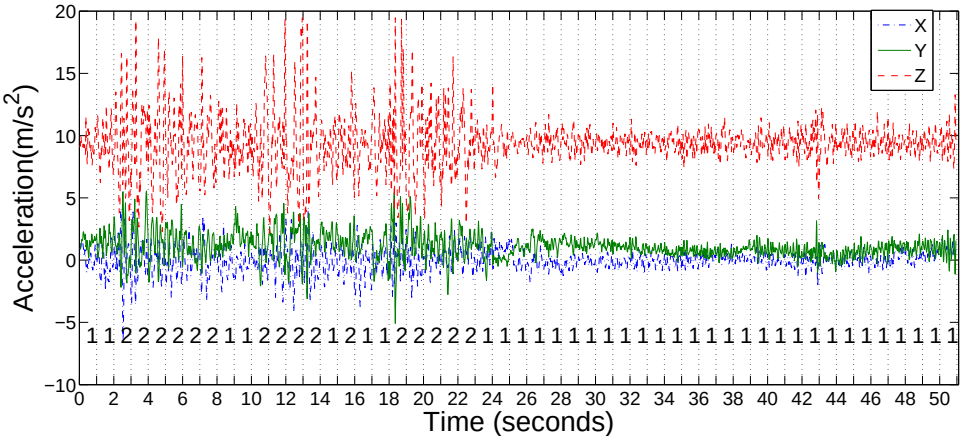



**Figure 4.1. Accelerometer data for bumpy(0-25s) and smooth(25-51s) road**

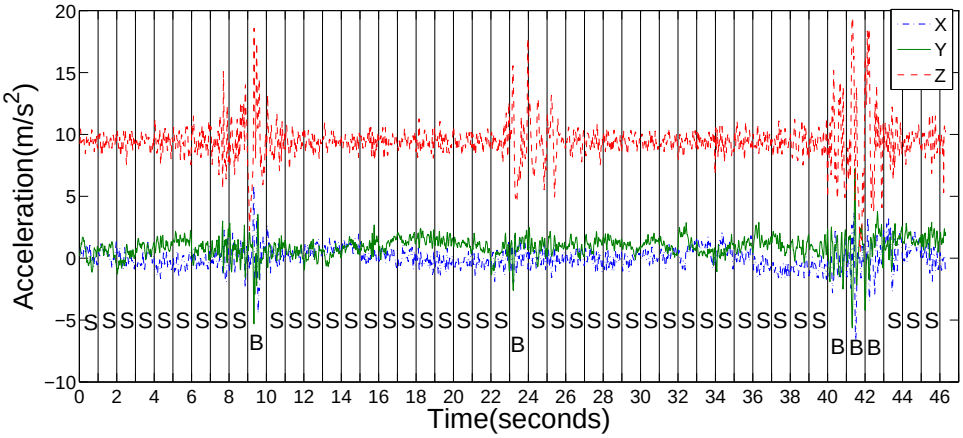



**Figure 4.2. Accelerometer Data for three speedbreakers**

road is the difference in the magnitude of variability of the accelerometer data within a window. In particular, the magnitude of variability of the Z axis captures the characteristic difference between the two classes of data. This feature is best captured by the standard deviation of the accelerometer data in the Z axis $(\sigma_Z)$. We tested this hypothesis by running our classification algorithm using only $\sigma_Z$ as a feature, and comparing it with the six-feature scenario. The classes predicted are exactly identical in both scenarios (They are marked in Figure 4.2). We thus conclude that the dominant feature deciding the bumpy/smooth characteristic is $\sigma_z$, and thus use it for our characterization. This observation, that the acceleration in Z direction is the dominant feature identifying bumps, is consistent with the observations of Pothole Patrol[10].

Let us explain the technique through an example. Figure 4.1 shows the reoriented data collected for a run that involved the vehicle moving on a rough, bumpy patch for 24 seconds, followed by moving on a smooth road for 23 seconds. As described above, this data is chunked into 1 second windows, features are extracted from it, and given as input to the K-means clustering algorithm. The (arbitrary) labels generated by the clustering algorithm are marked in Figure 4.1.

Now, the manual labeling is done over the clustered data. In this case label 2 corresponds to a bumpy road, and label 1 to a smooth road; such is fed to the SVM as labels for the training data. Now that the SVM is trained, it can be used to classify more data. Figure 4.2 shows 23 seconds worth of data corresponding to a smooth road, interspersed with solitary bumps, that were encounterd at roughly the $5^{th}$, $12^{th}$ and $21^{st}$ seconds. The SVM is used to then generate the classes for this data. The classes predicted by the SVM are marked in Figure 4.2. *S* is smooth and *B* is bumpy. Notice that the SVM correctly identifies all three bumps.

## 4.2 Vehicle Braking Detection

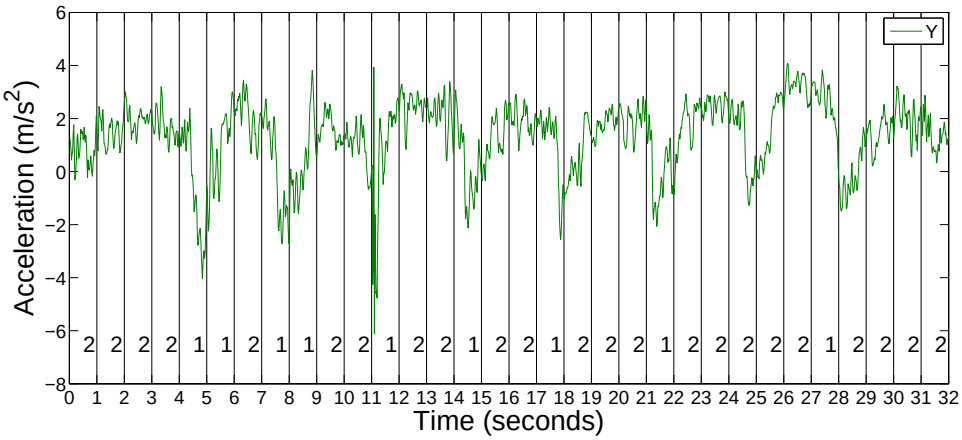



**Figure 4.3. Y axis Accelerometer for braking events(with labels)**

The problem we wish to address in this chapter is the detection of braking events. This has applications in traffic state prediction. For example, a number of braking events within a short interval of time on the same vehicle could indicate the presence of road congestion. A number of braking events observed at the same geographical location across vehicles could indicate some sort of obstacle on the road.

As in case of the bump detection algorithm, we use the reoriented accelerometer data, and chunk it into 1 second windows.

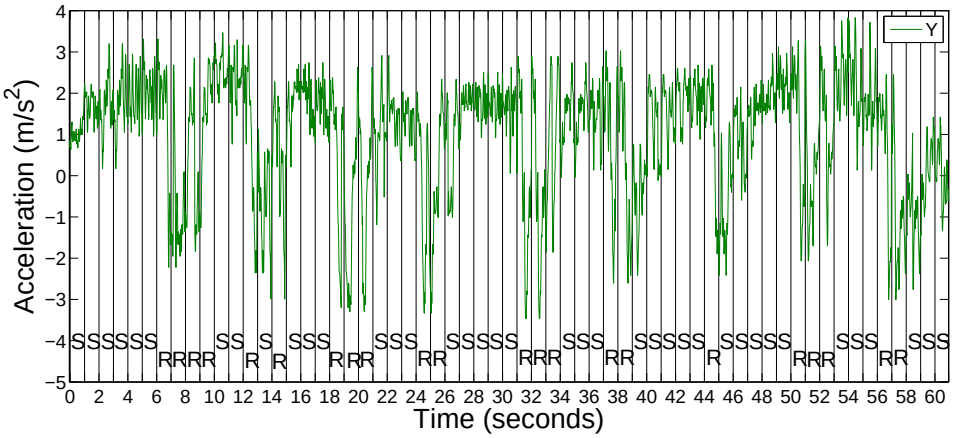



**Figure 4.4. Braking events with generated class labels**

The features used are different, though. A braking event lasts for around 60ms(See Figure 4.3). This is sudden, and much shorter than the duration of a bump event, which lasts for close to 500ms(Figure 4.2). Thus, the standard deviation $\sigma_Y$ is not able to classify the braking windows from the non-braking ones very well. For this reason, we use a different feature to help classify the braking data. We define a metric $\delta$ or difference metric defined as follows

$$\delta_X = max_{a_i \in window} a_i - min_{a_i \in window} a_i \tag{4.1}$$

Here $a_i$ is the acceleration recorded at the $i^{th}$ instant. (Recall that the accelerometer collects 50 readings per second) It is similarly defined for the Y and Z axes too. Thus now we now have nine features to choose from $\mu_X, \mu_Y, \mu_Z, \sigma_X, \sigma_Y, \sigma_Z, \delta_X, \delta_Y$ and $\delta_Z$. We found that the feature that captures the braking event the best, and thus is the most apt to be used for it is $\delta_Y$. Recall that Y is the direction of motion of the vehicle, and thus this is the axis in which the horizontal decceleration associated with braking can be best observed.

The technique used is identical to the one for bump detection, apart from the changed feature being used. On the extracted features, we use the K-means clustering algorithm, followed by a manual labeling stage, followed by training of a support vector machine. The classification phase then involves running the extracted features through the SVM. We shall not describe this in detail here, having already described the similar bump detection algorithm in chapter 4.1.

Figure 4.3 shows the reoriented accelerometer data for the Y axes for eight braking events over a 32 second time period, along with the labels generated by the K-means clustering algorithm. Here, we manually label the cluster 2 to be a non-braking event, and 1 to be a braking event

Figure 4.4 shows the reoriented Y axis acceleration for another run with multiple braking events. The data is 51 seconds long, with nine breaking events. The SVM is trained using the labels in Figure 4.3, and then the data in Figure 4.4 is classified using that SVM. The labels given by the classification algorithm are marked in the figure itself(S is smooth, R is braking).

## 4.3 Concluding Remarks

The classification phase of the technique we desribed above is a single pass algorithm. Thus it is appropriate to be used on a smartphone device, where space constraints might be significant. Even in the training phase, the data we store is only the features corresponding to each window, which comes out to be 16 bytes per second (2 float variables), or less than 1kb per minute - not large at all.

We end up using two kinds of features for classification $\delta$ and $\sigma$. We found $\sigma$ to be a much more robust parameter as compared to $\delta$. It is less susceptible to noise in the data. Thus it is used where possible - in our case for bump detection. When $\sigma$ is not a feasible feature, due to the small duration of the event, we use $\delta$. In case of high noise in the sensor data input, a filter to the accelerometer data may be necessary before using $\delta$. However, we leave this for future work.

# Chapter 5

# Enery Consumption Model

In this chapter, we build an energy consumption model for the Wolverine system. Further, we use this model to compare the energy used by our system to the Nericell system, and report the percentage savings in energy.

The points of energy consumption in our system are GPS, accelerometer, magnetometer and CPU(for computation). Previous work reports that the power consumption of CPU is lower than that of GPS and magnetometer by nearly 10 orders of magnitude [9]. Thus, we ignore the power consumed by CPU in our computation. The energy consumption by accelerometer and magnetometer depend on the rate of sampling. This remains constant throughout the algorithm, thus energy consumed depends only on the amount of time for which the sensor is kept on. [7] and [9] report that GPS energy consumption is largely independent of the received signal - neither the number of satellites, nor the signal strength, has any appreciable effect. Thus this too depends only on the amount of time for which GPS is kept on.

Just as Nericell, we too keep accelerometer and magnetometer on for 100% of the time. Thus, we can use the figures quoted in [14] to compute energy for these. Where our technique differs significantly from Nericell is in the use of GPS. Nericell uses GPS primarily for reorientation. The time for which GPS remains on can be represented by

$$\text{t} = \text{TTFF} + \text{Time till braking} + \text{Time for braking} \tag{5.1}$$

where TTFF is the time till first fix for GPS, time till braking is the expected time for which we must wait for a braking event to occur. If on an average, braking events occur once every $z$ seconds, this would be equal to $\frac{z}{2}$. Time for braking is the time for which the braking event is recorded. Wolverine too uses GPS primarily for reorientation. However, unlike Nericell, we need not wait for a braking event. We simply need to record the GPS values at two sufficiently far apart points. This can be done in two ways.

$$\text{t} = (\text{TTFF} + \text{Time to record}) \times 2 \tag{5.2}$$

In this method, we turn GPS on, record a value and turn it off. Then, when the vehicle has moved sufficiently, we turn on the GPS again and record another value of GPS.

$$\text{t} = \text{TTFF} + \text{Time to record} \times 2 + \text{Vehicle Motion Time} \tag{5.3}$$

In this method, we turn on GPS and record a location. Then GPS is kept on, as the vehicle moves. After it has moved a sufficient distance, the second reading is taken, and GPS is then turned off. Since we wish to minimize the amount of GPS usage time, we will use Equation 5.2 if TTFF $<$ Vehicle Motion Time and Equation 5.3 if it is the other way round.

According to [14], GPS uses $617.3mW$ of power, and runs for 10% of the total running time of the algorithm. The sensors and CPU together consume $31.85mW$ power, and run for 100% of the time. Our method saves energy by reducing the amount of time for which the GPS remains on.

Table 1 shows some values for the various parameters in the equations above. TTFF for GPS can be upto 5 minutes. However, we use network assisted GPS, which brings TTFF down to (Time till subsequent fix (TTSF) + Network Latency). Using these values in Equations 5.1 and 5.3 (since $VMT < TTFF$), GPS remains on for only 7.5 seconds $(5 + 0 \times 2 + 2)$ as compared to 67.5 seconds $(5 + 60 + 2)$. Thus, we get energy saving of 89% as compared to Nericell, per reorientation

| Activity | Time |
|---|---|
| TTFF | 5.5s |
| Time to Record | 0 |
| Time till braking | 60s |
| Time for braking | 2s |
| Vehicle Move Time | 2s |

**Table 5.1. The parameters of energy model**

event. Hence, in Wolverine, the GPS remains on for only 1.1% of the time, as compared to 10% for Nericell. Thus, the total energy saved, as compared to Nericell system is

$$1 - \frac{0.11 \times 617.3 \times 0.1 + 31.85}{617.3.1 + 31.85} = 58\%$$

Thus the Wolverine system consumes 58% lesser energy than Nericell, subject to certain assumptions on the parameters of the model.

# Chapter 6

# Evaluation

For data collection, we carried the phone on several rides at different locations in the IIT-Bombay campus on two different vehicles - a Suzuki Access 125 scooter, and a Bajaj autorickshaw. We performed the data collection in different road conditions, collecting readings from accelerometer, magnetometer and GPS. Our rides include roads which were bumpy, having speed-breakers and smooth. Additionally, we collected data with several braking and acceleration events, in order to test the braking detection technique.

While collecting the training data, we kept the mobile in a certain arbitrary orientation. We noted the number of events(braking or bump), and then used this knowledge while labeling. For test data, we collected data with the phone held in a different, arbitrary orientation. This enabled us to test both the reorietation and the machine learning algorithms. We applied reorientation algorithm on this data and fed the output to the machine learning algorithm, which labeled readings in each one second interval.

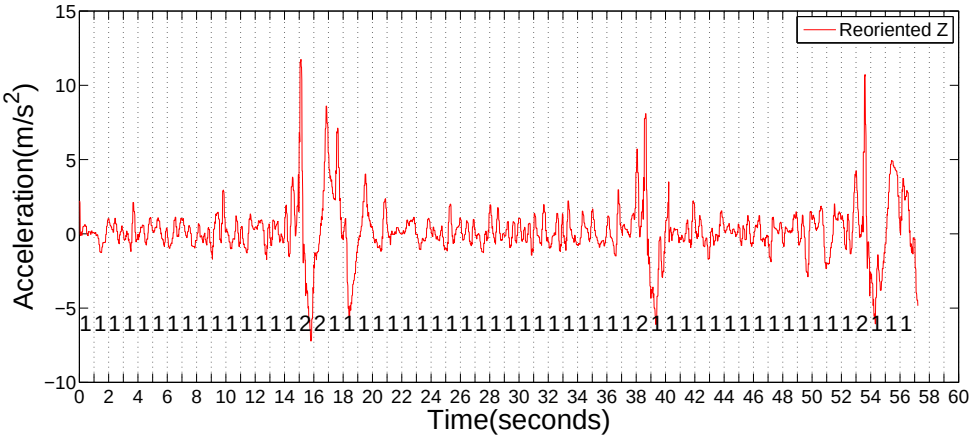



**Figure 6.1. Accelerometer readings in reoriented Z-direction for scooter training data, with clusters**

Figures 6.1 and 6.2 show the acceleration of the mobile phone in the reoriented Z direction, that is in the direction perpendicular to the plane of the vehicle, for experiments conducted on the Suzuki Access 125. We use the data in Figure 6.1 to train our system, and then validate it by testing on the data shown in Figure 6.2. The clustering algorithm generated labels, which are consistent with the ground truth (See clusters marked in Figure 6.1). The windows with cluster 2 correspond to three bumps - which were also noted during data collection. The points in cluster 1 are on a straight patch of road. After labelling the clusters as such, we proceed to train the SVM, and generate labels for the test data. Each *Bump* label generated by the algorithm is shown in the graph (we show the labels for windows within 2 seconds of each other as a single label. These correspond to the same bump event). The algorithm correctly identifies 18 out of the 20 bump events(all except those at the $170^{th}$ and $275^{th}$ second). No smooth road patch is incorrectly identified as a bump. Thus we get a *zero false positive rate*, and a *10% false negative rate*.

Figures 6.3 and 6.4 depict the acceleration in reoriented Z direction, for experiments conducted on the Bajaj autorickshaw. Again, Figure 6.3 is the training data, which is used to train a system which is validated on Figure 6.4. We observed 25 bumps during the run. However, the algorithm reports 27. In particular the bumps reported at the $477^{th}$ and $512^{th}$ second are false positives. Thus, in this run of the experiment, we get *zero fasle negative rate*, and a *8% false positive rate*.

Similarly, we plot the graphs for the experiments conducted for the braking detection algorithm. First, let us discuss the experiments done on the scooter. Figure 6.5 shows the training data, with the generated clusters marked. Figure 6.6 shows

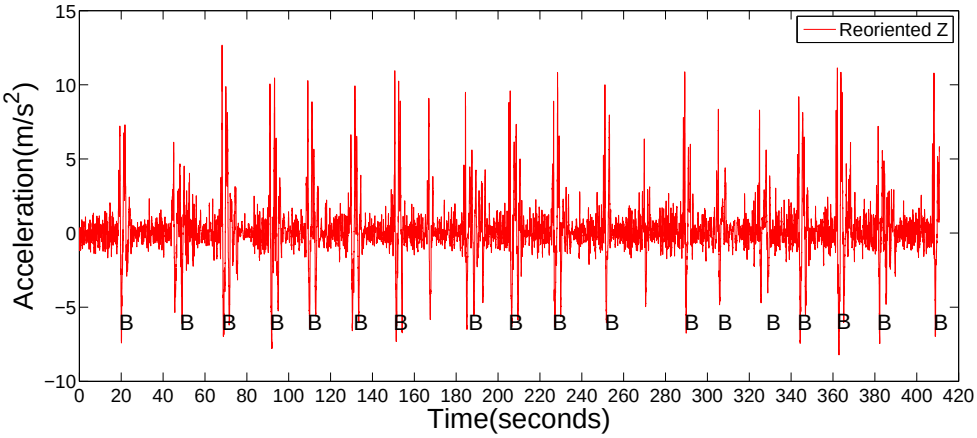



**Figure 6.2. Accelerometer readings in reoriented Z-direction for scooter test data, with labels**

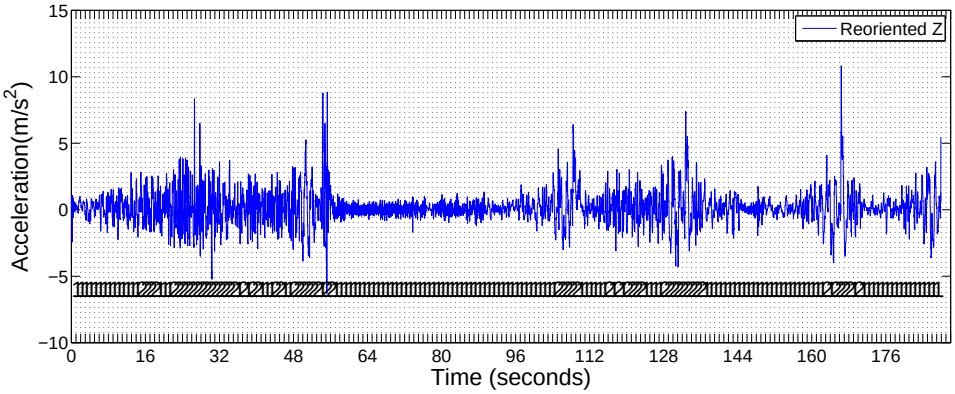



**Figure 6.3. Accelerometer readings in reoriented Z-direction for autorickshaw training data, with clusters**

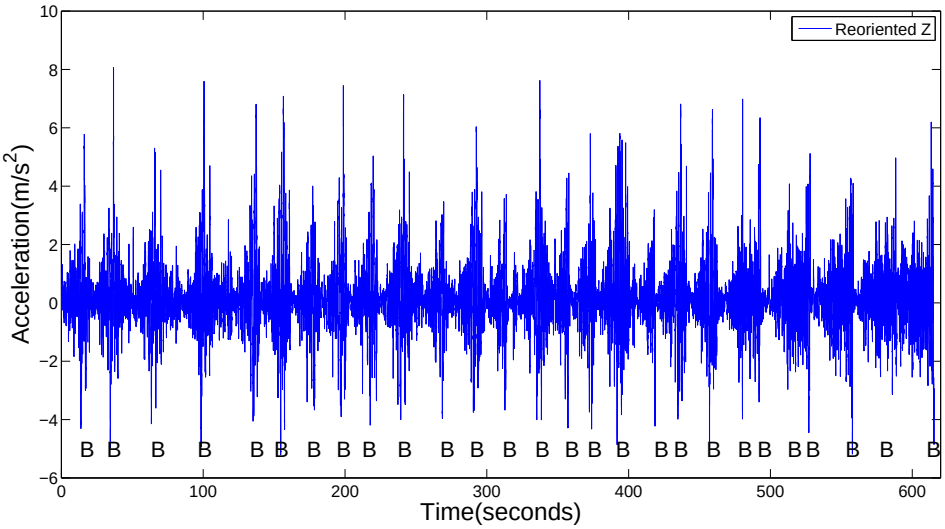



**Figure 6.4. Accelerometer readings in reoriented Z-direction for autorickshaw test data, with labels**

the test data, with the generated labels marked for braking events. Both figures show the acceleration in the reoriented Y direction, that is in the direction parallel to the motion of the vehicle. Out of a total of 37 braking events in the run, the algorithm correctly identifies 29, with one false positive (at the $200^{th}$ second). Thus, we get a false positive rate of 2.7% and a false negative rate of 21.6%.

Figures 6.7 and 6.8 show the same experiment conducted on a Bajaj autorickshaw. In this case, the system detect 33 out

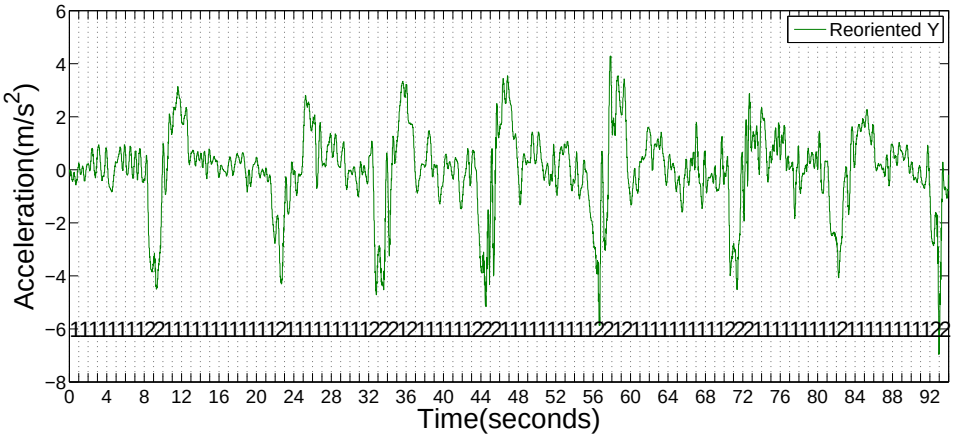



**Figure 6.5. Accelerometer readings in reoriented Y-direction for scooter training data, with clusters**

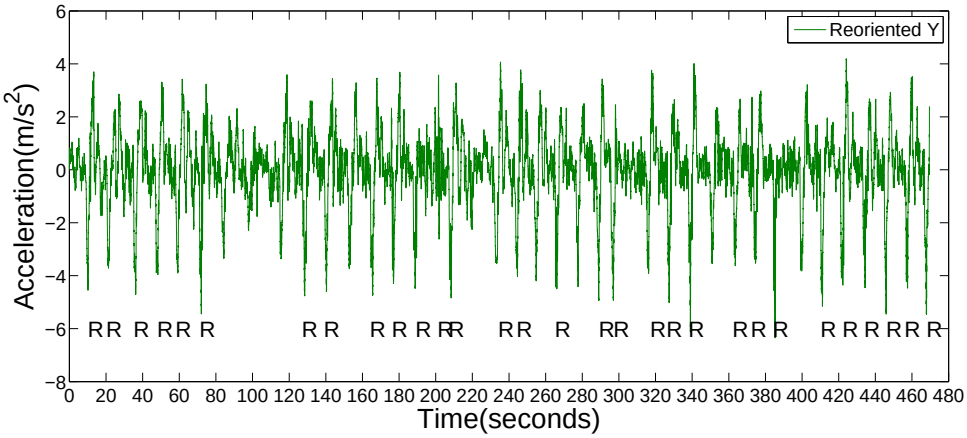



**Figure 6.6. Accelerometer readings in reoriented Y-direction for scooter test data, with labels**

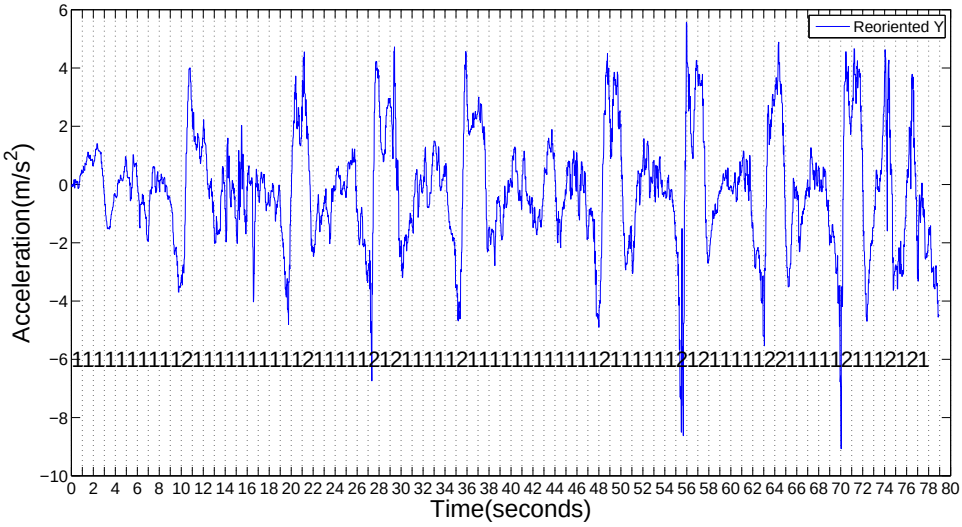



**Figure 6.7. Accelerometer readings in reoriented Y-direction for autorickshaw training data, with clusters**

of 38 braking events that we had recorded during the experiment. Thus, there are no false positives, and 5 false negatives, at the $36^{th}, 118^{th}, 150^{th}, 257^{th}, 282^{th}$ second. This gives us a false negative rate of 13.1%.

Both the reorientation algorithm and the vehicle state detection algorithms give good results, with the system giving low false positives and relatively low false negatives.

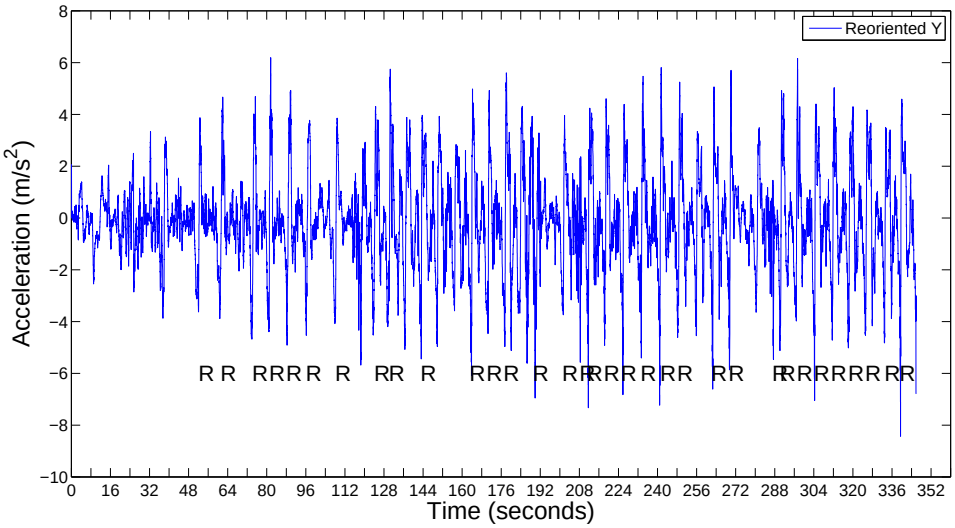



**Figure 6.8. Accelerometer readings in Y-direction after reorientation for autorickshaw test data, with labels**

# Chapter 7

# Conclusion and Future Work

As road traffic is increasing day by day, monitoring it in an effective way has been challenge to researchers. Since smart phones are penetrating into common people's lives very fast, utilizing the sensors available in them for traffic monitoring is good idea. The data processed by the mobile can be sent to a central server, which can use the information received to annotate maps accessed by the users through this application. This annotation can contain lot of information like the intensity of traffic at a junction, the bumpy nature of the road etc. All this can be done in an energy efficient manner by using low energy consuming components of the mobile like accelerometer and magnetometer and occasionally using GPS for localization and finding the bearing of the road. Also, applying machine learning techniques in classifying data can help the system to adapt to changing factors like nature of the road and vehicle type the users use. Some filtering techniques and advanced machine learning methods are to be applied to further improve the false negative rates of the Wolverine system.

# Bibliography


[1] Calculate distance, bearing and more between latitude/longitude points. http://www.movable-type.co.uk/scripts/latlong.html.

[2] Geomagnetic field. http://hi-android.info/src/android/hardware/GeomagneticField.java.html.

[3] K-means clustering. http://en.wikipedia.org/wiki/K-means_clustering.

[4] Right hand rule. http://en.wikipedia.org/wiki/Right-hand_rule.

[5] Sensor manager documentation. http://hi-android.info/src/android/hardware/SensorManager.java.html.

[6] Support vector machine. http://en.wikipedia.org/wiki/Support_vector_machine.

[7] M. Amir Yosef. Energy-aware location provider for the android platform. Master's thesis, University of Alexandria, Egypt, 2010.

[8] V. Bychkovsky, K. Chen, M. Goraczko, H. Hu, B. Hull, A. Miu, E. Shih, Y. Zhang, H. Balakrishnan, and S. Madden. The cartel mobile sensor computing system. In *SenSys'06*, pages 383–384, 2006.

[9] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[10] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan. The pothole patrol: Using a mobile sensor network for road surface monitoring. In *The Sixth Annual International conference on Mobile Systems, Applications and Services (MobiSys 2008)*, Breckenridge, U.S.A., June 2008.

[11] S. Guha, K. Plarre, D. Lissner, S. Mitra, B. Krishna, P. Dutta, and S. Kumar. Autowitness: locating and tracking stolen property while tolerating gps and radio outages. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 29–42, New York, NY, USA, 2010. ACM.

[12] H. Lu, W. Pan, N. D. Lane, T. Choudhury, and A. T. Campbell. Soundsense: scalable sound sensing for people-centric applications on mobile phones. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, MobiSys '09, pages 165–178, New York, NY, USA, 2009. ACM.

[13] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell. The jigsaw continuous sensing engine for mobile phone applications. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 71–84, New York, NY, USA, 2010. ACM.

[14] P. Mohan, V. N. Padmanabhan, and R. Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 323–336, New York, NY, USA, 2008. ACM.

[15] R. Sen, B. Raman, and P. Sharma. Horn-ok-please. In *MobiSys*, pages 137–150, 2010.

[16] R. Sen, P. Siriah, and B. Raman. Roadsoundsense: Acoustic sensing based road congestion monitoring in developing regions. In *SECON*, pages 125–133, 2011.

[17] J. Yoon, B. Noble, and M. Liu. Surface street traffic estimation. In *MobiSys 07*, pages 220–232. ACM, 2007.