The Jigsaw Continuous Sensing Engine for Mobile Phone Applications

Hong Lu,[†] Jun Yang,[*] Zhigang Liu,[*] Nicholas D. Lane,[†] Tanzeem Choudhury,[†] Andrew T. Campbell[†]
[†]Dartmouth College, {hong,niclane,campbell}@cs.dartmouth.edu, {tanzeem.choudhury}@dartmouth.edu
[*]Nokia Research Center, {jun.8.yang,zhigang.c.liu}@nokia.com

Abstract

Supporting continuous sensing applications on mobile phones is challenging because of the resource demands of long-term sensing, inference and communication algorithms. We present the design, implementation and evaluation of the *Jigsaw continuous sensing engine*, which balances the performance needs of the application and the resource demands of continuous sensing on the phone. Jigsaw comprises a set of sensing pipelines for the accelerometer, microphone and GPS sensors, which are built in a plug and play manner to support: i) resilient accelerometer data processing, which allows inferences to be robust to different phone hardware, orientation and body positions; ii) smart admission control and on-demand processing for the microphone and accelerometer data, which adaptively throttles the depth and sophistication of sensing pipelines when the input data is low quality or uninformative; and iii) adaptive pipeline processing, which judiciously triggers power hungry pipeline stages (e.g., sampling the GPS) taking into account the mobility and behavioral patterns of the user to drive down energy costs. We implement and evaluate Jigsaw on the Nokia N95 and the Apple iPhone, two popular smartphone platforms, to demonstrate its capability to recognize user activities and perform long term GPS tracking in an energy-efficient manner.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems

General Terms

Algorithms, Design, Human Factors, Performance

Keywords

Mobile Phone Sensing, Machine Learning, Activity Recognition, Power Management

1 Introduction

Today's mobile phones come equipped with an increasing range of sensing, computational, storage and communication resources enabling *continuous sensing applications* to emerge across a wide variety of applications areas, such as, personal healthcare, environmental monitoring and social networks. A key challenge of continuous sensing on mobile phones is to process raw sensor data from multiple sensors (e.g., accelerometer, microphone, GPS, gyroscope, digital compass, camera) and compute higher level inferences and representations of human activities and context – possibly in real-time and communicate these higher level inferences to the cloud. Early examples of continuous sensing applications for the phone are emerging. UbiFit [7] uses accelerometer data to recognize human activities, monitoring the amount of exercise by an individual and using an unobtrusive ambient display on the phone to encourage appropriate changes in levels of exercise. PEIR [19] uses inferences such as the user's transportation modes to produce personalized environmental impact reports that track how the actions of individuals affect their exposure and contribution to environmental problems such as carbon emissions. CenceMe [16] uses the phone's accelerometer and microphone to infer the user's activity and social interactions and update their sensing presence on online social networks such a Facebook and MySpace.

Continuous sensing applications require careful resource management in order to facilitate long periods of data collection and processing. For example, data should be able to be collected for a complete recharge cycle, while leaving enough energy for the phone to still operate as a phone – that is, to make calls, text, read email, surf the web. Mobile phone sensing also requires that the inferences are robust to various mobile phone context. For example, classifiers must be able to withstand a person placing the phone at different body positions and cope with a wide variety of noisy sensor inputs that occur when people use their phones in different real-world scenarios. Sensors react differently under different conditions, for example, the microphone is robust to the phone being placed at different body positions while the accelerometer is not. In addition, each phone sensor has specific tradeoffs largely based on the nature of the data it samples. For example, accelerometer data is fairly inexpensive to process, compared to the microphone data which typically has a much higher sampling rate and is computational costly to analyze. Recently, researchers have been challenged to build mobile phone sensing systems that are both robust and resource efficient [16,29]. To date, the majority of mobile phone sensing systems are stovepipe solutions which attempt to address these challenges but in a very application specific way.

In this paper, we present *Jigsaw*, a continuous sensing engine for mobile phone applications which require continuous monitoring of human activities and context. By developing a reusable sensing engine and proposing application agnostic techniques, we allow Jigsaw to be both resilient and energy-efficient. It uses sensor-specific *pipelines* that have been designed to cope with the individual challenges presented by each sensor. The techniques we develop are not tied to any specific application but are based on studying the specific problems that arise in mobile phone sensors and phone usage patterns.

Jigsaw operates entirely on mobile phones and does not require the use of an external server to perform any part of its operation. Jigsaw implements the following techniques as part of its continuous sensing engine on the Apple iPhone and Nokia N95: i) resilient accelerometer data processing allows inferences to be robust to different phone hardware, orientation, and body positions; ii) smart admission control and on-demand processing for the microphone and accelerometer data adaptively throttle the depth and sophistication of the pipelines when the input data is low quality or uninformative, and iii) expensive pipeline stages (e.g., audio activity classification, sampling from the GPS) are triggered judiciously and are adaptive to the human behavioral patterns. More specifically, new approaches to calibration of the accelerometer, classification of activities that are independent to the position of the phone on the body, and filtering of extraneous activities and movements are proposed as part of the accelerometer pipeline. The microphone pipeline proposes new approaches that drive down the computational cost of sensing and classification of sound activities while sacrificing as little accuracy as possible. Techniques embedded in the microphone pipeline reduce redundant classification when the sound type does not change over certain timescales and short circuits pipeline computation for common but distinctive classes of sound that are observed. Finally, we design a smart GPS pipeline that uses learning techniques and drives the duty cycle taking into account the activity of the user, energy budget, and duration over which the application needs to operate – the pipeline automatically regulates the sampling rate to minimize the localization error. We believe the flexibility and adaptability of Jigsaw makes it suitable for a wide range of emerging continuous sensing applications for mobile phones.

The remainder of the paper has the following structure. Section 2 describes the challenges and related work of robust and energy-efficient classification using the three sensors (accelerometer, microphone, and GPS). In Section 3, we describe the Jigsaw architecture and algorithms. Section 4 provides our prototype implementation. Section 5 presents a detailed evaluation of the system. Section 6 provides two proof-of-concept sensing applications built on top of the Jigsaw engine. And Section 7 concludes.

2 Design Considerations

In this section, we discuss the design considerations that underpin the development of Jigsaw's sensing, processing, and classification pipelines for three of the most common sensors found on mobile phones today; that is, accelerometer, microphone and GPS.

2.1 The Accelerometer Pipeline

The energy cost of sampling the accelerometer is not prohibitive and neither is the required computation. The only technical barrier to performing continuous sensing using the accelerometer is the robustness of inferences.

Figure 1 demonstrates the difficulty in achieving robustness with accelerometer inferences. In this experiment, the time series output of an accelerometer-based activity classifier is shown, which is trained from data collected from a phone carried in the front pocket of a pair of jeans. Initially, the phone is in the user's pocket while the person is cycling, and the inferences are fairly accurate. Midway through the experiment the user receives a phone call, causing noticeable but understandable temporary misclassification. After the phone call, the user puts the phone in his backpack. For the remainder of the experiment the classifier performance is significantly degraded. Figure 2 helps explain the result of Figure 1 in more detail. It shows sampling during cycling from the accelerometer when the phone is in the user's backpack and front pocket. Note, when the phone is in the backpack the raw data simply captures artifacts of vibration, while the data from the front pocket clearly shows a cyclic pattern. If a classifier is trained with data from one body position, it will struggle to recognize the activity when the phone is placed in other body positions different from where the training data is sourced.

This experiment demonstrates two distinct types of the robustness problem, one associated with the body position and the other associated with errors that occur during temporary states, such as, taking the phone out of pocket, which interrupt an on-going activity (e.g., cycling). We refer to these temporary states as *extraneous activities*, which we more formally defined as activities driven by user interactions with the phone that are unimportant to the application performance such as texting or making a phone call. Despite the large amount of prior work that uses accelerometers for physical activity recognition [5, 11, 23, 24], none developed techniques to counter extraneous activities, which degrade the robustness of inference when performing continuous sensing on mobile phones. In addition, prior work does not address pipeline techniques to cope with changing body positions, although some (e.g., [5,16]) researchers have measured the negative effects of changing position.

The Jigsaw accelerometer pipeline counters extraneous activities by recognizing: i) periods of user interaction with the phone, (e.g., a user texting); and ii) transition states such as standing up or picking up the phone. Jigsaw does not require the phone to maintain any particular body position and inference accuracy remains consistently high even when the body position changes, as discussed in Section 5.1. Jigsaw addresses this challenge by using i) orientation independent features rather than only magnitude that is used in exist-

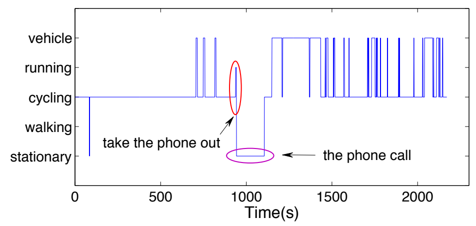


Figure 1: Activity inferences are inaccurate when the phone is in a backpack and the model was trained for a front pocket.

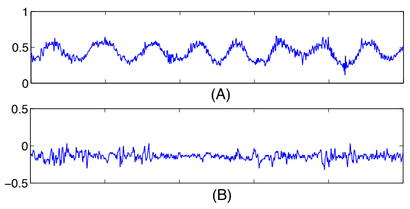


Figure 2: Accelerometer samples of cycling while the phone is (A) in the pant pocket, (B) in the backpack

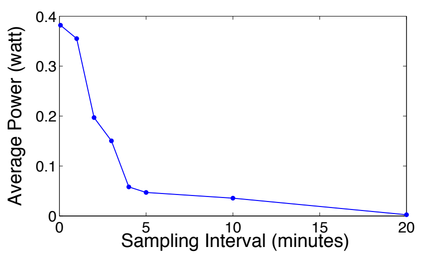


Figure 3: GPS Power Profiling of N95

ing systems such as [16,24]; ii) one-time device calibration, which can be transparent to the user; and iii) classification techniques where activities are modeled by splitting them into several sub-classes, each of which is tied to particular body positions; for example, the same activity class (e.g., cycling) is partitioned based on body positions (e.g., a class for upper body positions and a class for lower body positions). Collectively, in a natural way these design techniques allows the user to use the phone in natural way because Jigsaw is able to intelligently adapt itself to different context.

2.2 The Microphone Pipeline

The difficulties of continuous sensing with the microphone pipeline are due to resource efficiency. The microphone generates data at a much higher sampling rate than other sensors and places a heavy burden on the computational resources of the phone. We demonstrate the high CPU burden of a microphone pipeline by performing an experiment on an Apple iPhone using a relatively low audio sampling rate of 8 kHz. We first perform a baseline measurement and find the average CPU load when playing a 256kbps AAC music file is 8% with a range of 3-15%. In comparison, the CPU load for signal processing and feature extraction is 6% on average. During classification using a Gaussian Mixture Models (GMM) classifier, the CPU usage rises to 22%, assuming 10 different sound types are classified. The CPU load increases as the number of sound classes increase; for example, when 20 different sound classes are supported the CPU load is 36%. The number of classes supported by a single phone will likely rise as applications become more sophisticated or when multiple applications use the Jigsaw engine.

It is clear that these CPU usage numbers are too high for a continuous background process which needs to have minimal impact on the primary operations of the mobile phone. Existing microphone sensing systems [22, 26] avoid CPU overhead by relying on remote servers; however, these systems still can not perform continuous sensing due to the energy burden of data transmission between the phone and the cloud. While [16] performs classification on the phone, it only detects one kind of sound – a single class. The Jigsaw microphone pipeline is influenced by our prior work on SoundSense [14]. However, SoundSense does not support a classification pipeline for continuous sensing, rather, it explicitly focuses on the challenge of personalizing a generic sound classification algorithm based on monitoring user behavior over time. In contrast to SoundSense, the Jigsaw microphone pipeline does not attempt to learn new sounds. Rather, it focuses on reducing the CPU load while sacrificing as little robustness as possible. SoundSense only needs to run a single audio classifier in the background whereas the Jigsaw engine needs to support three pipeline processes in parallel presenting a more demanding low power design environment. The Jigsaw engine reduces the computation of the microphone pipeline with an admission control and duty cycle component that regulate the amount of data that enters the microphone pipeline. The duty cycle is on-demand rather than fixed, as in previous work [16,29]. Within the sound inference stage of the pipeline two techniques are used: i) early pipeline short-circuiting for common but distinctive classes of sound; and ii) minimizing redundant classification operations when the sound type does not changed.

2.3 The GPS Pipeline

It is well-known that continuous sensing using GPS is costly in terms of energy consumed due to sampling the GPS [8, 10, 16]; for example, if we assume that the energy budget of GPS is 25% [8] of the phone's battery, the GPS can only offer continuous GPS sensing for 2 hours using the Nokia N95. It is possible to lower the sampling rate and increase battery life to an acceptable level, but this sacrifices sensing resolution. At low sampling rates the robustness of the GPS pipeline (i.e., the accuracy of the location estimates) starts to become highly sensitive to the mobility pattern of the user. Accuracy becomes unpredictable in this case; for example, the error associated with the same low sampling rate can vary wildly from person to person depending on their respective mobility patterns. Even for the same person the error can vary dramatically due to changing mobility pattens during a normal day.

The key technical challenge in performing continuous sensing with the GPS pipeline is determining an optimal schedule of sampling the GPS which minimizes the localization error when possible. A common approach to this problem is to attempt to use a static duty cycle [16] or use precomputed duty cycles for different situations [28,29]. However, no single duty cycle can be appropriate for all people and devices, e.g., the best sampling schedule for the GPS for a graduate student who spends most of the day sitting in a lab is very different from a UPS worker who drives around a city all day long. Another common approach is to make the duty cycle adaptive by making runtime duty cycle schedule [10] based on the sensed GPS data, or, using other sensors such as the accelerometer [1, 18] to trigger GPS sampling when motion is detected. However, these approaches still neglect a number of important aspects that impact a schedule such as the battery budget, hardware differences, or the actual required timing and duration for location estimates from the application.

The Jigsaw GPS pipeline overcomes the limitation of the high sampling cost of the GPS by *learning* an adaptive sampling schedule using a Markov Decision Process (MDP) [9]. Jigsaw automatically adapts the GPS sampling schedule to minimize the expected localization error. The sampling schedule is adaptive to the mobility mode of the user by leveraging real-time classification of activity. Using movement detected by the accelerometer to switch the GPS sensor on/off has been proven to be effective. Jigsaw uses the user's mobility mode to achieve finer grain control of the GPS sampling. The sampling schedule is also adaptive to the duration over which the application requires GPS sensing, the remaining battery budget, and time lapse from the start of an application. The Jigsaw GPS sampling strategy optimizes the specific combination of mobility, sensing duration, and hardware status (e.g., remaining battery budget) and adapts when any of those factors changes. Note, that although the current version of Jigsaw focuses on the GPS sensor, the technique that Jigsaw uses is agnostic to the type of location sensor and is therefore applicable to other types of localization methods, such as, WiFi triangulation.

3 Jigsaw Detailed Design

In this section we describe the detailed design of the Jigsaw continuous sensing engine. The Jigsaw architecture comprises three classification pipelines that sit between the application layer and hardware sensors; these are, the accelerometer, microphone and GPS pipelines which are responsible for sampling, processing and classification of data from the sensors. More specifically, the accelerometer and microphone pipelines provide streams of inferences and the GPS pipeline provides location estimates for applications built on Jigsaw. Note, Jigsaw is designed to support one or more applications at the same time on a mobile phone. Applications can choose to use one or all of the pipelines depending on the applications needs.

3.1 Accelerometer Pipeline Design

Figure 4 provides an overview of the processing stages for the accelerometer pipeline. Raw accelerometer data is broken into frames. If necessary, a one-off calibration process is initially applied before preprocessing occurs. During preprocessing there are a number of internal stages, beginning with normalization which converts the raw readings into gravitational units (i.e., G) using device-specific parameters learned during calibration. Normalized accelerometer data is processed by admission control, where extraneous movements of the phone are efficiently filtered out. We define extraneous movements as transition movements that are unimportant to the application performance such as taking the phone out of a pocket or standing up. Admitted frames are passed to the projection stage, which translates data into an orientation independent global coordinate system making any subsequent processing insensitive to the phone orientation. The final output of projection is fed to the feature extraction stage. The extracted feature vector is then provided to the activity classification stage which recognizes five common physical activities; these are, stationary, walking, cycling, running, and

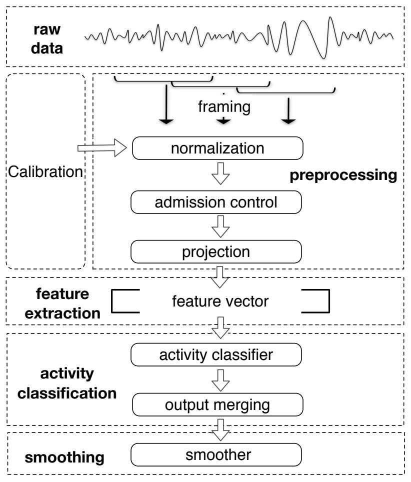


Figure 4: Jigsaw Accelerometer Pipeline

vehicle (i.e., car, bus). The activity classification stage uses a split-and-merge technique to handle the different phone body placement positions [21]. Finally, the stream of inferences are then fed through a smoothing step. We now discuss each of the stages in turn.

3.1.1 Calibration Stage

Calibration is a one-time process that determines offset and scaling factors (i.e., sensitivity). The offset and scaling factors are parameters required by the normalization stage, which compensates for hardware variation and converts raw samples into the standard G unit. Naive calibration methods involve pointing each axis of the device strictly up and down to get the necessary positive and negative 1g readings. However, it is difficult for untrained users to accurately perform this cumbersome and error-prone procedure. The Jigsaw one-time calibration process can be either user driven or fully transparent to the user. In the case of *user driven calibration*, the user is asked to hold the phone still in several different directions which are not necessarily aligned with gravity. The whole process usually takes less than one minute. In the case of *automatic calibration*, the phone opportunistically collects accelerometer samples whenever the phone is determined to be stationary, for example, when the user is sitting, taking photos, or the phone is placed on a table, etc. Once sufficient samples are collected Jigsaw performs the same computation as the user driven calibration to estimate the normalization parameters. During automatic calibration the phone can be used normally by the user. Based on our experience a comprehensive automatic calibration usually takes between one and two days, as discussed in Section 5. We now describe the Jigsaw calibration computation in detail.

Let $\vec{a} = (a_x, a_y, a_z)$ be a raw accelerometer reading, and $\vec{g} = (g_x, g_y, g_z)$ be the actual acceleration along each axis in G unit. Let K_x, K_y, K_z and b_x, b_y, b_z be the respective scaling factors and offsets of the accelerometer. Subsequently,

$$g_{axis} = K_{axis} \cdot a_{axis} + b_{axis}, \text{where } axis = x, y, z.$$

and we define a target function of:

$$f(K_x, K_y, K_z, b_x, b_y, b_z) \triangleq \sqrt{g_x^2 + g_y^2 + g_z^2}$$

If the accelerometer is stationary, $f(\cdot) = 1$. A solution to this parameter estimation problem is a least square estimator based on linear approximation of function $f(\cdot)$ [13]. However, the method in [13] needs pre-calibration knowledge about the sensor and uses a computationally intensive recursive procedure. Without loss of generality, we set the initial guess of parameters at $(K, K, K, 0, 0, 0)$, which means all the scaling factors are K and offset is 0, since by design the sensitivity for all axes are approximately the same and the offsets close to zero. After taking linear approximation of $f(\cdot)$ around the initial point $(K, K, K, 0, 0, 0)$ using Taylor expansion, we obtain a linear equation,

$$f(\cdot) \approx \frac{a_x^2}{\|\vec{a}\|} \cdot K_x + \frac{a_y^2}{\|\vec{a}\|} \cdot K_y + \frac{a_z^2}{\|\vec{a}\|} \cdot K_z + \frac{a_x}{\|\vec{a}\|} \cdot b_x + \frac{a_y}{\|\vec{a}\|} \cdot b_y + \frac{a_z}{\|\vec{a}\|} \cdot b_z = 1$$

Note that the above equation is not affected by the value of K. Once a sufficient number of $\vec{a}$ (i.e., accelerometer readings sampled under stationary conditions) are obtained, the parameters can be estimated by solving an over-determined system using the linear least square estimator.

$$\begin{aligned} & \left[\ \hat{K}_x \quad \hat{K}_y \quad \hat{K}_z \quad \hat{b}_x \quad \hat{b}_y \quad \hat{b}_z\ \right] \\ = & \left[\ 1 \quad 1 \quad 1 \quad \ldots \quad 1\ \right] \cdot A \cdot (A^T A)^{-1} \end{aligned}$$

where N is the total number of static samples, $[1\ 1\ \ldots\ 1]$ is an all-one row vector of length N, and A is the coefficient matrix. Let $\vec{a}_i = (a_{x,i}, a_{y,i}, a_{z,i})$ (i = 1, 2, ..., N) be the ith static accelerometer reading. The ith row vector of matrix A is calculated by $\left[\frac{a_{x,i}^2}{\|\vec{a}_i\|}, \frac{a_{y,i}^2}{\|\vec{a}_i\|}, \frac{a_{z,i}^2}{\|\vec{a}_i\|}, \frac{a_{x,i}}{\|\vec{a}_i\|}, \frac{a_{y,i}}{\|\vec{a}_i\|}, \frac{a_{z,i}}{\|\vec{a}_i\|}\right]$.

The linear least square estimator gives the optimal estimate of the parameters. Error in this estimation will diminish with additional stationary readings made available. A stationary state detector is used to select qualified $\vec{a}$ (i.e., stationary accelerometer readings). Our stationary detector begins by dividing raw data into candidate frames each contain M successive samples. For each frame, the mean and standard deviation are calculated for the three axes. If all three standard deviations fall below a percentage threshold σ of the mean, we assume the device is stationary and the mean of the frame (m_x, m_y, m_z) is treated as a qualified $\vec{a}$. The frame length M and threshold σ control the quality of the candidates. Longer M and tighter σ will generate higher quality candidates but less often. When qualified, the candidates usually only show constant gravity. However, there are rare exceptions, e.g., the phone is in free fall, or in a constantly accelerating vehicle. In these cases, the magnitude of the mean (m_x, m_y, m_z) will be different from the majority good candidates. Such outliers are filtered out.

3.1.2 Preprocessing Stage

Preprocessing takes a raw stream of accelerometer readings and produces data that is projected into an coordinate system that is independent of the mobile phone orientation, filtering out extraneous activities and movements. These two internal steps of filtering and projection within the preprocessing stage result in increased robustness in accelerometer inferences. The extraneous activities and movements filtered out include: i) periods of user interaction with the phone, (e.g., inputing a text message or browsing the web), and ii) transition states, such as, standing up or picking up the phone. Preprocessing applies the following internal steps in the following order: normalization, admission control and projection.

Normalization. Initially, accelerometer readings are used to estimate the vertical direction (gravity) in the local coordinate system. In [17], Mizell shows the mean of accelerometer readings along each axis over a period of time is a good estimate of the gravity direction. The same approach is used here to estimate the gravity for each frame (128 samples, about 4 seconds) of accelerometer readings. The frame length of 128 samples offers a good tradeoff between estimation accuracy and latency. Each sample is converted to unit G using the offset and sensitivity parameters provided by the calibration process. The output of normalization is normalized frame $\vec{a}_i = (x_i, y_i, z_i), i = 1, 2, \ldots, 128$. Its gravity estimation is denoted by $\vec{g} = (m_{\mathbf{x}}, m_{\mathbf{y}}, m_{\mathbf{z}})$, where $m_{\mathbf{x}}, m_{\mathbf{y}}$, and $m_{\mathbf{z}}$ are means of their respective axes.

Admission Control. Extraneous activities and movements are filtered from the incoming stream of frames using different detection strategies. Extraneous movements (i.e., transition states) will alter the orientation of the accelerometer with respect to gravity. Detection of these states requires that Jigsaw tracks D, the absolute difference between two successive frame's gravity estimation, $D = \|\vec{g}_{new} - \vec{g}_{last}\|$. It captures the change of sensor orientation. When D exceeds a predefined value, θ_d, the frame is suspected to be an extraneous movement event and skipped. The detection of extraneous activities (i.e., user interaction), in contrast, is based on Jigsaw monitoring mobile phone key presses or GUI events. Frames are filtered out when any of these events occur during sampling.

Projection. Projecting incoming data on to a global coordinate system makes feature extraction insensitive to changes in phone orientation. The projection on to a global vertical (the gravity direction) and horizontal (perpendicular to the gravity) directions is performed by Jigsaw as follows: Let $\vec{v}_i$ and $\vec{h}_i$ denote the components in the global vertical and horizontal direction. We know $\vec{h}_i$ lies on the horizontal plane which is orthogonal to gravity. However, without a compass it is impossible to get the absolute direction of $\vec{h}_i$ using the accelerometer alone. We can calculate the length of the $\vec{v}_i$ (denoted by v_i) by the dot product, $v_i = \vec{a}_i \cdot \vec{g}$ with sign of v_i indicating the direction. Using v_i and $\vec{g}$ allows us to compute $\vec{h}_i$ simply by using $\vec{v}_i = v_i \times \vec{g}$, $\quad \vec{h}_i = \vec{a}_i - \vec{v}_i$. Because the direction of $\vec{h}_i$ is not informative, we simply use its magnitude $\|\vec{h}_i\|$, as the measure of horizontal movement in our final coordinate space. These algorithmic calculations are repeated for each $\vec{a}_i$ such that the output of preprocessing is $\{(v_i, \|\vec{h}_i\|), i = 1, 2, \ldots, 128\}$, which is passed to the feature extraction stage.

3.1.3 Feature Extraction Stage

There are 24 accelerometer features used in Jigsaw, which present a combination of time-domain and frequency-domain features, summarized in Table 1. Three time-domain features are computed: mean, variance and mean-crossing rate. In the frequency domain, all features are based on spectrum analysis. Different activities have different energy

Time domain	mean, variance, mean crossing rate
Frequency domain	spectrum peak, sub-band energy, sub-band energy ratio, spectral entropy

Table 1: Motion Feature Set

stationary	
stationary	sitting, standing, on the table
walking	
lower body	front and back pants pockets
in hand	in hand when reading the screen, armband, in hand when swinging naturally
all others	jacket pocket, backpack, belt
cycling	
lower body	front and back pants pockets
upper body	jacket pocket, armband, backpack, belt
running	
running	all body postions
vehicle	
vehicle	car, bus, light rail

Table 2: Activity Subclasses

distributions over the frequency spectrum. Walking usually peaks around 1Hz, running at $2-3$Hz, whereas in vehicle samples often contain more energy in high frequency bands due to the vibration of the journey. The sampling rate of the Nokia N95 accelerometer is about 32Hz. According to the Nyquist sampling theorem, it can capture signal frequency characteristics up to 16Hz. Based on these observations, we carefully design the frequency features to differentiate the target activities. We use the peak frequency as a feature. The peak's location indicates the dominant frequency of the activity. The spectral entropy is a rough description of the frequency distribution. If the distribution is flat, the spectral entropy is high; if the distribution is peaky, its value is low. Finally, spectrum sub-band energy is calculated on four frequency sub-bands: $B_1(0,1], B_2(1,3], B_3(3,5], B_4(5,16]$(Hz). Three sub-band energy ratios, i.e., the ratio between B_1 and B_2, B_3 and B_4, $B_1 \cup B_2$ and $B_3 \cup B_4$ are used to summarize the energy distribution in low and high frequency ranges. All these features are extracted from the projected data $(v_i, \|\vec{h}_i\|)$, 12 features for each direction. Collectively, these features form the 24-dimension feature vector used for activity classification.

3.1.4 Activity Classification Stage

Jigsaw partitions single activity classes into separate activity sub-classes according to body positions (see Table 2). This addresses the challenge presented by different body position and improves inference robustness, as discussed in Section 2.1. For example, the walking and cycling activities, which suffer noticeably from the body position issue are decomposed into three and two subclasses, respectively. The split technique structures the training data set and then builds the classifiers offline accordingly.

The output merge step happens at run time. The phone performs inference based on the trained sub-class models and merges the sub-class labels back to the original semantic labels for the output. Although internally the classifier is body-position sensitive, the final result hides this. For example, the activity of cycling with the phone in your pants pocket (lower body) or in a backpack (upper body) are classified separately; however, they both will report cycling. Since the difference in performance of classification models is insignificant (see Section 5.1), we choose the decision tree classifier for its efficiency. We generate a tree of depth 7 using J48 learning algorithm provided by Weka [30], as discussed in Section 5.1. The last stage of the accelerometer pipeline applies a lightweight sliding window smoother to the classification output to filter out outliers.

3.2 Microphone Pipeline Design

The Jigsaw microphone pipeline is shown in Figure 5. The stream of audio data from the microphone is divided into frames by the preprocessing stage. During preprocessing, the internal steps of admission control and duty cycling are applied to dynamically regulate the resources used by the pipeline. Following this the feature extraction stage extracts a combination of features depending on the pipeline workflow; that is, either specific selected features for recognizing human voice or more general purpose Mel-Frequency Cepstral Coefficients (MFCC) [32] features. The computational bottleneck in the microphone pipeline is the activity classification stage, which uses GMM (Gaussian Mixture Model) based classification to classify a number of different sound types required by applications. This inference step is computationally intensive and shapes the remainder of the pipeline design. Two techniques are built into the pipeline design to compensate for this bottleneck. Jigsaw uses a voice classification stage to determine whether a frame contains common and easily identified sound classes using a very efficient yet sufficiently accurate decision tree classifier. If the frame is human voice then the activity classification stage is not required, saving pipeline computational resources. Otherwise, the frame is forwarded to the activity classification stage. An internal similarity detector step in the activity classification stage eliminates redundant classification. It reduces the workload when adjacent frames are of the same sound class. It should be noted that this occurs frequently, for example, when someone drives a car, a large number of frames of the driving class go through the audio pipeline consecutively. In this case, the similarity detector compares the incoming frame and previously classified frames based on similarity of their features. If they are similar enough, the prior label is directly used. The final stage is smoothing. We now discuss each of these stages in turn.

3.2.1 Preprocessing Stage

The primary responsibility of preprocessing is to regulate the resource usage of the microphone pipeline. Frames are rejected if, for example, the phone is too muffled to capture sound. In addition, the duty cycle of microphone sampling is adapted based on the ambient environment. We now discuss the three preprocessing steps; that is the framing, admission control and duty cycling steps.

Framing. Preprocessing operates on a frame by frame basis with each frame containing around 64 ms of audio or 512 samples when sampling at the phone standard frequency of 8kHz. Previous audio recognition work [6, 15] uses overlapped frames of 25-32 ms in order to capture subtle changes at the expense of computation. Jigsaw takes a different approach where frames are longer and do not over-

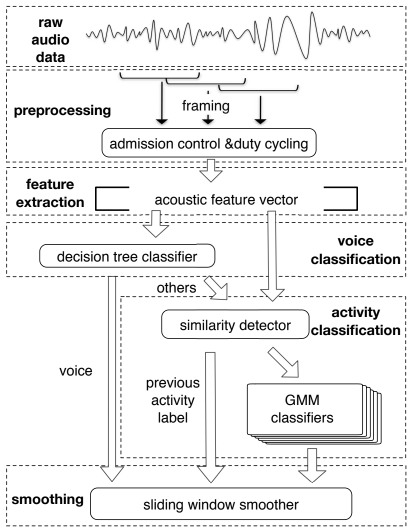


Figure 5: Jigsaw Microphone Pipeline

lap with each other. This significantly reduces the quantity of frames required to be processed. Because the goal is to recognize sound classes associated with human activities, a longer frame length is useful to capture low frequency components of activities.

Admission Control. Large chunks of silence or low volume frames often occur between sound activities. These uninformative silences, which Jigsaw is designed to skip, usually are caused by a lack of sound events (e.g., it is late at night, the user is away from the phone) or by an undesirable phone context (e.g., muffled in the backpack or being located at some distance from a sound source). The Jigsaw frame-based admission control uses the intensity of the frame to determine if a frame should be admitted or not. The intensity is measured by the root mean square (RMS) [25], which is a lightweight time domain feature. The use of intensity relies upon thresholds which are chosen empirically for different phones due to the different sensitivity of the built-in microphones. In practice, the Nokia N95 records higher volume audio than the iPhone 3G under the same conditions. In real-world scenarios, once a daily activity begins there is a strong tendency for it to continue. Therefore, if a frame passes admission control, all subsequent frames are accepted for the next three seconds. There are common cases where there is no acoustic event captured for a long period of time. When this occurs, duty cycling of the microphone is applied.

Duty Cycling. During the periods of no acoustic events, the optimal duty cycle of the microphone pipeline depends on the application. However, a low duty cycle, such as one frame per second (64 ms out of 1000 ms), is sufficient for detecting many everyday activities. Jigsaw adapts the duty cycle to save phone resources based on the behavior of admission control and optionally accelerometer based activity inferences. Duty cycle adaption is achieved through decaying the sampling frequency (i.e., increasing the sampling interval), when no acoustic event is captured. Examples of such scenarios include: when the user is sleeping, in a quiet environment, or the phone is in a locker. The sampling rate begins to increase once the level of human activity increases (e.g., the person wakes up and picks up the phone for use). A minimum duty cycle bounds how far the duty cycle is able to be reduced and can be set by the applications.

3.2.2 Feature Extraction Stage

Jigsaw computes a variety of acoustic features, as shown in Table 3. In the temporal domain, the *low-energy frame rate* is used. All other extracted features are in the frequency domain. To ensure the robustness of the microphone pipeline under different conditions, the spectrum is normalized and its DC component is removed before frequency analysis. Therefore, the computed features are less sensitive to the volume of the audio sample and thus reduce the significance of the differences in microphone sensitivity and phone position. Features are computed only on-demand, so those frames which do not reach, for example, the activity classification stage (e.g., frames that contain voice) will not have any of the additional activity classification specific features computed. Those features used by the voice classification stage are computed with low-overhead yet have sufficient discriminative power to distinguish human voice. A frame that does not contain human voice will be processed by the activity classification stage. In this case, more powerful and computationally expensive features are required – a 13 dimension Mel-Frequency Cepstral Coefficient (MFCC) feature vector is computed which reuses the spectrum computed earlier during the voice feature computation. Classification is not, however, performed directly on these frame based features. In Jigsaw, the unit for classification is not a single frame, which would be costly and unnecessary. Given the goal of classifying sounds associated with human activities, which are likely to last at least for tens of seconds or more, the unit of classification is a 20 frame classification window (i.e., 1.28s). The low-energy frame rate is the ratio of the frames whose energy are below 50% of the average energy of all the frames in the classification window. The mean and variance of each frame-based frequency feature for all frames in the window are also used as features for classification.

3.2.3 Voice Classification Stage

Other than silence, which is filtered during the admission control stage of preprocessing (see Section 3.2.1), perhaps the next most common class of sound people encounter is human voice. Voice classification is designed to perform class-specific binary classification which recognizes classification windows that contain human voices accurately and inexpensively in terms of computational resources. This significantly lowers the typical overhead of executing the Jigsaw microphone pipeline, because voice is no longer processed by the activity classifier, which is computationally expensive given it performs general purpose classification and tries to pattern match potentially dozens of different activity sounds. The acoustic characteristics of voice are fairly unique [6] relative to other classes of sound associated with specific human activities. The voice classification stage uses a discriminative approach that relies on a decision tree classifier. We train the decision tree using the WEKA workbench [30] based on features specifically selected for the classification task, as shown in Table 3. Although the classifier consumes very

Category	Feature set
voice	Spectral Rolloff [12],Spectral Flux [25] Bandwidth [12],Spectral Centroid [12] Relative Spectral Entropy [6] Low Energy Frame Rate [25]
other activities	13 MFCC coefficient feature set [32] Spectral Centroid [12],Bandwidth [12] Relative Spectral Entropy [6] Spectral Rolloff [12]

Table 3: Acoustic Feature Set

Category	Activies	Note
Voice	reading, meeting, chatting, conference talks, lectures,	10 female, 22 male, 40 clips
Target sounds	street noise in city&high way, crowd noise of job fair&party washing,brushing teeth vacuuming, shower, typing	from multiple scenarios 119 clips
Other sounds	walking, rain, wind, microwave climbing stairs, toilet flushing music, fan, plane, dish washer elevator,clapping, white noise	83 clips

Table 4: Description of the Data Set

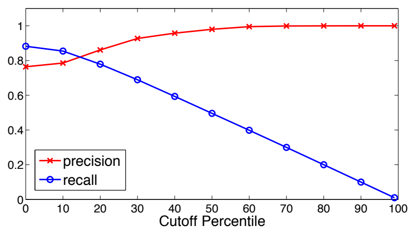


Figure 6: Precision and Recall v.s. Cutoff Threshold

little resources to execute, it achieves classification accuracy (see Section 5) comparable to other classifiers [27,31]. The sound data set we use is collected sporadically over a nine month period using both the Nokia N95 and the Apple iPhone. All the recordings are in mono, 8kHz, 16bit PCM format. The sound categories are shown in Table 4. The total amount of data is about 1GB. The data set is divided into two non-overlapped sets, i.e., a training set (80%) for training models and a test set (20%) for evaluation purposes.

3.2.4 *Activity Classification Stage*

The activity classification stage of the Jigsaw pipeline recognizes a diverse set of sound classes that are associated with human activities (e.g., driving, washing hands). The current Jigsaw prototype system detects the following sound classes - each of which is closely related to a common activity: brushing teeth, shower, typing, vacuuming, washing hands, crowd noise, and street noise. However, the classification stage is costly in terms of computation which increases linearly with the number of sound classes being classified. Therefore, the activity classification stage is a potential bottleneck in the microphone pipeline, unless it is engaged infrequently. The internal design of the activity classification stage uses a similarity detector to curb how often the complete classification logic is used. In what follows, we discuss how classification is done using a GMM classifier. We also discuss how we limit the need to perform GMM classification using our similarity detector.

GMM Classifiers. Jigsaw classifies sound using a naive Bayes classifier with equal priors. The likelihoods are estimated using Gaussian Mixture models (GMM), one for each activity. GMMs are widely used in audio processing and have proved to be effective [15,27]. The class with highest likelihood is statistically the most likely activity. However, if the highest likelihood is too small, lower than the cut-off thresholds θ of a class, the system still rejects it and reports the sound as unknown (i.e., other sound). We use K-means initialized expectation-maximization (EM) for training. The K-means initialization makes the expectation-maximization converge faster with more stability. Jigsaw supports both full covariance matrices and simplified diagonal covariance matrices for the Gaussian components of the GMM. The later uses fewer parameters and reduces the computational complexity at the cost of accuracy. A comparison of these two approaches is discussed in Section 5. The likelihood threshold θ_m of a particular sound class m is estimated using the percentile cut-off of the likelihood of the training data. It differentiates the sample that belongs to that category and other unknown sounds. A higher cut-off percentile benefits the precision of the classifier and decrease its recall and vice versa. An appropriate tradeoff depends on application's specific needs; for example, Figure 6 shows the change in precision and recall for the vacuuming sound using different cutoff percentiles. We choose the cutoff at the 10th percentile to balance precision and recall.

Similarity Detector. Human activities tend to have durations on the order of ten seconds or more. Some activities, such as, driving, walking down the street, taking a shower last considerably longer. During these activities classifying every frame is redundant because successive frames in the stream of audio data are similar and all the classification logic is repeated with the arrival of each new frame. The similarity detector is designed to eliminate this redundancy, allowing Jigsaw to dramatically reduce the workload of the activity classification stage. The similarity detector exploits the fact that frames from the same activity have similar feature vectors. By computing a similarity measurement, the classification process can be skipped if the distance in this metric is small enough. Once an activity sound is recognized by the GMM classifier, the similarity detector stores the activity label and the feature vector $\vec{x}_{act}$. When the successive feature vector $\vec{x}_{new}$ arrives, it measures similarity using the cosine distance,

$$distance = 1 - \frac{\vec{x}_{act} \cdot \vec{x}_{new}}{||\vec{x}_{act}||\,||\vec{x}_{new}||}$$

If the distance is below a predefined threshold δ, it indicates that the previous inference is still on-going. The previous activity label is directly dispatched to the smoothing stage (see below), otherwise, the classification operation is trigged and the activity label and $\vec{x}_{act}$ are updated. By changing the similarity threshold δ, we can control the rate at which GMM classification is avoided. However, error may increase if the similarity threshold is loosened too much. Keeping a longer history of previously classified feature vectors can be beneficial when the user switches back and forth between several activities. However, in practice, even when storing only one pattern, the similarity detector is able to save about 70% of the classification operations with a moderate 5% accuracy penalty, as discussed in Section 5. Using the similarity detector, Jigsaw substitutes costly classification with a light-weight distance measurement. Note that as the number of sound classes increases, the classification cost grows linearly, but the cost of distance calculation remains unchanged.

3.2.5 *Smoothing Stage*

Classification results can be noisy for a variety of reasons; for example, the user is interacting with the device, there is a sudden change in the ambient background sound, or simply due to the dynamic nature of sound. A smoothing step is therefore helpful in removing outliers. There are sophisticated smoothing methods for audio processing, such as the Hidden Markov Models (HMMs) used in [14]. However, these models are computationally demanding if the total number of sound categories needed by applications is large, and they need to be retrained every time a new sound category is added. As an alternative approach, we apply a very simple sliding window smoother on the classification output. It is computationally lightweight, insensitive to the number of the activity classes, and still provides a performance boost, as discussed in Section 5.

3.3 GPS Pipeline Design

Jigsaw uses the inferences obtained from the accelerometer pipeline to provide a real-time adaptive GPS duty cycle based on the actual mobility patterns of the user. The basic intuition here is that in order to reduce the energy consumption and keep a low localization error, activities associated with faster speed need a higher GPS sampling rate and vice versa. We model this problem as a discrete-time Markov Decision Process (MDP) to learn the optimal GPS duty cycle scheme. We encode the following attributes in the model: the GPS battery budget, an expectation of the duration that the application tracks a user, and the user's mobility pattern. The goal of the optimization process is to learn the best duty cycle policy for spending the given battery budget over the tracking duration according to the user's runtime mobility level provided by the accelerometer pipeline and the remaining battery budget level. We now describe our MDP formulation:

The Markov decision process [9] is represented as a four-tuple (S,A,P,R), where S is the state space, A is the action space (i.e., different GPS duty cycles), $P_a(s,s^{'}) = \Pr(s_j = s^{'}|s_i = s, a_i = a)$ is the transition probability that executing an action a in state s leads to the next state $s^{'}$, and $R(a,s)$ is the reward function for performing action a in state s. A policy π is a mapping from S to A that determines an action for every state $s \in S$. The quality of a policy is indicated by the expected sum of total future rewards, which are discounted to ensure convergence. γ is the discount rate and satisfies $0 < \gamma \leq 1$. The value of a state s under policy π is the expected sum of the discounted rewards by following policy π from s, defined as,

$$V_\pi(s) = R(a,s) + \gamma \sum_{s'} P_{\pi(s)}(s,s')V_\pi(s').$$

If all the parameters in the MDP are known, the optimal policy is computed by solving the Bellman Equation:

$$V(s) = \max_a \left(R(a,s) + \gamma \sum_{s'} P_a(s,s')V(s')\right)$$

The Bellman equation can be solved using the Policy Iteration algorithm [9].

In our model, the composite state is,

$$s_i = (m_{accel}(i), e_{gps}(i), t_i), 1 \leq t_i \leq T$$

where m_{accel} is the inference from the Jigsaw accelerometer pipeline, e_{gps} is the remaining GPS energy budget, t is the current time index (time tick in the total sensing duration), and T is the total number of time ticks. In the above discrete-time model, the total amount of time T_s (seconds) is quantized into T time ticks evenly such that each time tick takes $t_s = T_s/T$ seconds. The time tick is set as a dimension of the composite state so that the learned policy changes according to time index, because the same battery level at a different time point could mean a totally different thing, e.g. 50% battery left is really bad in the early morning, but is abundant in the early evening. The time tick transition is strictly increasing by one in each tick and modeled as a degenerate Markov chain,

$$P(t_j|t_i) = \begin{cases} 1 & \text{if } t_i = t_j = T \\ 1 & \text{if } t_j = t_i + 1 \\ 0 & \text{otherwise} \end{cases}$$

The activity inference m_{accel} takes four values based on the motion speed: 1=stationary, 2=walking, 3=running or cycling, and 4=in vehicle. The state transition probabilities,

$$\{P(m_{accel}(j) = k|m_{accel}(i) = l), 1 \leq k,l \leq 4\}$$

can be learned from activity traces. We distinguish workdays and weekends and use two distinct transition matrixes, because the user's mobility patterns are typically different during workdays and weekends.

The action space A is a set of predefined GPS sampling rates. We use six sampling actions

$$A = \{a|\text{sample every } interval(a) \text{ time units}, 1 \leq a \leq 6\}.$$

where $interval(a)$ is the sampling interval. The GPS receiver usually takes a while to lock on to a satellite signal when switched on, and it has a significant turn-off lag (e.g., the power-off delay for GPS is about 30 seconds on Nokia N95 [10]). Thus, the power consumption of a sampling interval lower than 45 seconds is almost identical to continuous sampling. Therefore, we use 6 sampling rates. For a = 1, 2 ..., 6, the sampling interval, $interval(a)$, is 20min, 10min, 5min, 2min, 1min, 5s (GPS always on), respectively. An action with a higher index uses a faster sampling rate and thus consumes more energy. $e_{gps}(i)$ is the amount of remaining GPS energy budget. We divide the total energy budget E(Joule) assigned to GPS sensing into L levels linearly and thus each energy level contains $E_0 = E/L$ (Joule). Let $\{\text{power}(a), a \in A\}$ be GPS energy consuming rates (power) over different sampling intervals $\{interval(a), 1 \leq a \leq 6\}$, which are obtained from energy profiling. The transition probability of the energy consumed by GPS sensing from level l to $l+1$ are calculated as:

$$p(a) = \text{power}(a) \cdot t_s/E_0, \quad a \in A.$$

The probability $p(a)$ is proportionally higher for faster sampling rates and lower for longer sampling interval. We model the transition probability of $e_{gps}(i)$ as a function of GPS sampling rate $a \in A$, such as,

$$P_a(e(j)|e(i)) = \begin{cases} 1 & \text{if } e(i) = e(j) = 1 \\ 1 - p(a) & \text{if } e(i) = e(j) = l, l \neq 1 \\ p(a) & \text{if } e(j) = e(i) - 1 \\ 0 & \text{otherwise} \end{cases}$$

where 'gps' subscript for $e(\cdot)$ is omitted. The activity inference $m_{accel}(i)$, the energy state $e_{gps}(i)$, and time tick t_i are independent to each other. Therefore, the overall composite state transition probability can be defined as,

$$P_a(s_i, s_j) = P\left(m_{accel}(j) | m_{accel}(i)\right) \cdot P_a\left(e_{gps}(j) | e_{gps}(i)\right) \cdot P\left(t_j | t_i\right)$$

The reward function depends on the sampling action and the activity inference. There is a penalty if the budget depletes before the required duration T. We define

$$R\left(\left[\left(m_{accel}(i), e_{gps}(i), t_i\right), a\right]\right) = \begin{cases} -R_{nopower} & \text{if } e_{gps}(i) = 1,\ t_i < T \\ k_m \cdot f\left(m_{accel}(i), a\right) & \text{otherwise} \end{cases},$$

where k_m is a reward adjustment coefficient depending on the motion speed and $f\left(m_{accel}(i), a\right)$ is the reward of taking action a when the user is in activity inference m_{accel}, usually defined as,

$$f\left(m_{accel}(i), a\right) = m_{accel}(i) \cdot a (1 \leq a \leq 6).$$

The reward function reflects the tradeoff between energy and localization error. A higher duty cycle is more accurate and generates more rewards in one time tick, particularly when the m_{accel} is also high. However, it also drains the battery quickly, if the battery budget runs out before the required sensing duration with all the following steps suffering penalties. The model is design to maximize the total reward. Greedily maximizing current reward is unwise. The MDP learning algorithm finds the optimal balance between the current action reward and potential future reward according to the knowledge of the user's activity inference transition pattern. Once the system is modeled, the policy iteration algorithm [9] is applied to learn the statistically optimal duty cycling policy. The output is a three dimensional table. For each state, denoted by a tuple (activity inference, remaining energy budget level, time tick), there is a corresponding sampling rate policy. The table is stored on the mobile phone. Using the table is straightforward and computationally lightweight. During runtime Jigsaw keeps track of the time tick, remaining energy budget, and the activity inference from the accelerometer pipeline. It then does a table lookup to find the optimal duty cycle for GPS sampling. This is an efficient duty cycle mechanism that is tailored to the mobile phone and adaptive to the user's behavior.

4 Implementation

We validate the Jigsaw design with prototype implementations on the Apple iPhone and the Nokia N95. The signal processing and classification algorithms are approximately 2000 lines of C code. Other components (i.e., GUI, sensing) are written in the native languages for each platform, specifically Objective C [2] for the iPhone and Symbian C++ [20] for the N95. The software architecture of the Jigsaw implementation is shown in Figure 7. Jigsaw uses a 32Hz sampling rate for the accelerometer and streams 8 kHz, 16-bit, mono audio from the microphone. The Nokia N95 provides direct access to the GPS sensor, but the iPhone uses a hybrid localization scheme using WiFi triangulation and GPS, and developers have no way to select the sensor via APIs. Consequently we benchmark and evaluate the Jigsaw GPS pipeline primarily on the N95. Jigsaw is implemented as a background service for the Nokia phone, such that multiple applications can use it at the same time. Under the current iPhone SDK [3], creating a daemon service is not yet allowed, therefore Jigsaw is implemented as a library for the iPhone. As shown in Figure 7, sensing and processing are in separated threads. The processing threads awake only when data is ready. Circular buffers and semaphores ensure continuous and asynchronous operations. On the iPhone, threads are all standard POSIX threads. On the N95, all processing threads are Symbian threads, and the sensing threads are actually Symbian Active Objects [20] for efficiency.

The Jigsaw system is extensible by design. The APIs allow access to different pipeline components for applications to directly use individual components or to add new features/classifiers beyond the default set. Currently the Jigsaw system exposes three basic sets of APIs to access the output of three main stages of classification pipelines, i.e., the preprocessing, the feature extraction, and the classification results. For example, our demo application Greensaw (see Section 6) utilizes the accelerometer pipeline for calorie expenditure calculation. The application first examines the classification result. If the activity is stationary or in vehicle, then only the resting energy expenditure is accumulated. If the activity is walking or running, data acquired from the preprocessing API is fed into a step counter, which is used to compute the activity calorie consumption combining with the user's height and weight. There are two classifier options for the accelerometer, a decision tree classifier and a multivariate gaussian classifier (see Section 5.1). For audio pipeline, both the full covariance GMM and diagonal covariance GMM are supported. Model parameters are stored in configuration files to allow customization. For example, application developers can add labels and parameters (i.e., mean, covariance and cut-off threshold) of GMM models to support new sound classes.

We design the Jigsaw prototype to optimize for CPU usage at the expense of a larger memory footprint. When it is possible we pre-compute values offline and store these as files on the phone. For example, for the audio classification pipeline, the GMM models directly use the inverse and determinant of covariance matrix as the parameters. We compute Σ^{-1} and $|\Sigma|$ offline instead of using Σ itself. Table 5 shows the runtime benchmark of the accelerometer pipeline. It takes approximately 5ms to process 4 seconds of accelerometer data. The CPU usage for the iPhone (with a simple GUI) is about 0.9% $\sim$ 3.7%. For the Nokia N95, the CPU usage is about 1$\sim$ 3% and the power consumption is about 38mw measured using the Nokia energy profiler [20]. Table 6 shows the runtime benchmark for the audio classification pipeline. The GMM results represent the total runtime for 7 activity GMMs in our prototype implementation. If a sound event is detected, the frame based feature extraction takes approximately 3.5ms for each admitted frame. The window based features are extracted in the 1.28s classification window and then classification takes place. Even when the full pipeline is engaged, every 1.28s sound signals only take approximately 100ms to process. When active, the CPU usage is approximately 6% $\sim$ 15% for iPhone and 7% $\sim$ 17% for Nokia N95 and dependent on the workflow of the pipeline. The power consumption on the N95 under this condition is approximately 160mw.

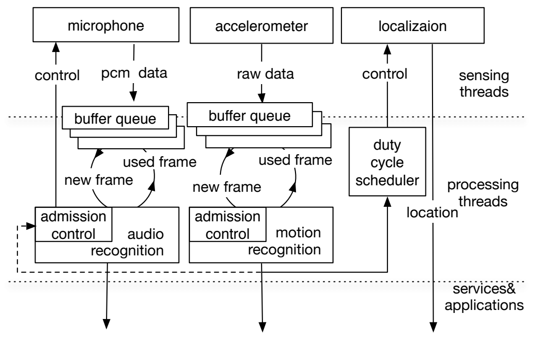


Figure 7: Jigsaw Software Architecture

Stage	time(ms) iPhone	time(ms) N95
normalization	0.19	0.16
admission control	0.10	0.08
projection	0.53	0.65
feature extraction	3.56	4.21
classification	<0.01	<0.01

Table 5: Accelerometer Runtime Benchmark

Stage	time(ms) iPhone	time(ms) N95
(i) sound detection	0.41	0.36
(ii) feature extraction	3.29	3.44
(iii) voice classification	0.01	0.01
(iv) similarity measure	0.02	0.02
(iv) classification with full cov GMM	10.04	10.43
(iv) classification with diagonal cov GMM	4.55	4.76

Table 6: Audio Runtime Benchmark

	Accuracy w/o Split&Merge(%)				Accuracy with Split&Merge(%)			
	DT	MG	SVM	NB	DT	MG	SVM	NB
cycling	82.62	82.41	86.60	77.45	92.05	90.07	92.88	90.87
vehicle	92.87	93.80	93.52	83.59	90.52	87.47	90.29	89.83
running	98.11	97.18	97.40	98.37	98.01	97.40	98.03	97.30
stationary	94.25	96.81	97.48	94.99	95.19	98.07	97.68	96.19
walking	90.35	91.89	93.90	88.55	96.81	97.04	96.66	95.17
Average	91.64	92.42	93.78	88.59	94.52	94.01	95.10	93.87

Table 7: Classifier Accuracy with and w/o Split and Merge

	lab	manual	automatic
$Scale_x$	305.10	303.88	303.78
$Scale_y$	299.60	295.48	299.44
$Scale_z$	296.42	295.48	297.08
$Offset_x$	2.06	13.63	24.50
$Offset_y$	2.70	14.47	25.54
$Offset_z$	3.70	16.88	26.64

Table 8: Parameter Estimated by Three Calibration Methods

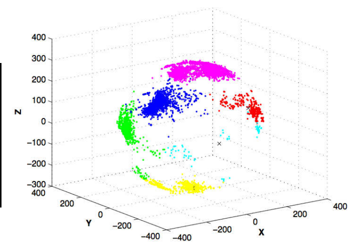


Figure 8: Auto Calibration Samples

5 Evaluation

In this section, we present the detailed performance evaluation of the Jigsaw pipelines. For the accelerometer, we collect a data set from 16 people including 12 males and 4 females. For each subject, the data is collected from multiple sessions in natural settings. No researcher follows the subject during the data collection. Each participant is asked to carry multiple phones in different body positions and annotate the beginning and the end of the activities they perform. A separate phone works as a remote to control all the sensing phones via bluetooth and collect user inputs. The data set is split into two parts, half of the data set is used as the training set and the other half as the test set. For audio, we use the test data set introduced in Section 3.2.3. We benchmark the GPS pipeline on the Nokia N95, because it efficiently supports the background process and the Nokia energy profiler utility is convenient and accurate.

5.1 Accelerometer Pipeline Results

Table 8 shows the calibration parameters carefully obtained during: i) a controlled lab experiment, ii) user driven calibration, and iii) automatic calibration. In user driven calibration, we use a 2 second frame length and the threshold $\sigma = 2\%$. The user is requested to generate 12 samples of different orientations. For automatic calibration, a N95 is carried for 1 day and the threshold is set to $\sigma = 3\%$. The candidate samples collected by the automatic method are visualized in Figure 8. The samples scatter around the sphere with 1G radius and roughly cover all 3 axes, which ensure the high quality of estimated parameters. The user manual calibration that requires the user's attention outperforms the automatic one. We test both calibration methods over a 80 sample test set containing only static readings of 1G in different orientations. For N95, the error is up to 1.1% for the manual calibration and up to 2% for the automatic one. The average calibration errors are 0.55% and 0.76%, respectively.

actual\output	voice	other
voice	0.8535	0.1465
other	0.0408	0.9592

Table 9: Confusion Matrix for the Voice Classifier

We repeat the automatic calibration experiment with Apple iPhone 3G, the average error is 0.58%, slightly better than N95. In practice, the iPhone accelerometer data is not as noisy. Moreover, the N95 only has a few applications using the device in landscape mode, resulting in less candidate samples in that orientation. For the iPhone, the user generates more qualified data in the landscape mode when browsing the web or using other applications.

We evaluate the classification accuracy with and without the split-and-merge process, as shown in Table 7. Four commonly used classifiers, Decision Tree (DT), Multivariate Gaussian Model (MG), Support Vector Machine (SVM), and Naive Bayes (NB) are compared. For all the classifiers, the split-and-merge process increases the average accuracy, particularly, for the Naive Bayes classifier. The major accuracy improvement is seen in the cycling and walking activities, which are divided into multiple subclasses, as shown in Table 2. Although our overall result slightly underperforms previous work (e.g. [24] reports 95% accuracy), we use only the accelerometer sensor (GPS is used in [24]) and place no restriction on the device orientation or body position.

5.2 Microphone Pipeline Results

Table 9 shows the confusion matrix of the first level voice classifier. A three classification result (3.84s) smoothing window is applied. The recall of voice frames is 85.35%. Although it is not as high as the 90% achieved by more sophisticated and demanding voice detectors, e.g., [6], it still performs well considering its lower resource consumption. The performance comparison of the full covariance matrix GMM model and diagonal covariance matrix GMM model over the seven activities is shown in Figure 10 and Figure 11. Gener-

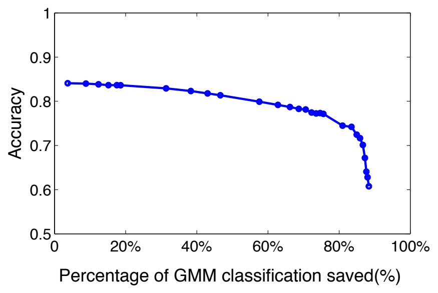


Figure 9: Tradeoff between the Accuracy and Percentage of GMM Classification Saved

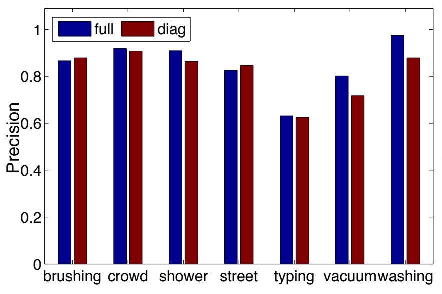


Figure 10: Precision of Two Types of GMM

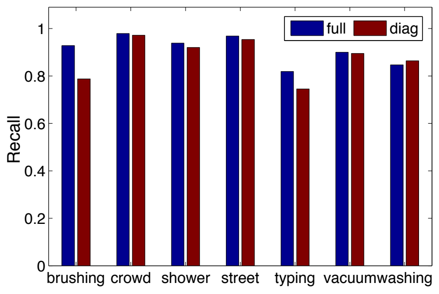


Figure 11: Recall of Two Types of GMM

ally speaking, the full covariance GMM model slightly outperforms the diagonal covariance one in both metrics with an average improvement of 2.98% for precision and 3.47% for recall. However, its runtime is more than double the diagonal one which is more suitable for mobile phones. The precision of the typing activity classification is approximately 60% for both classifiers. The typing sound, for example, is more dynamic than other activity sounds. The pace and intensity of key strokes vary greatly even for the same person, making it hard to be characterized accurately by the GMM model. The model misclassified typing in some cases, for example, when in fact the phone carried in the pocket or backpack hits against keys or other objects.

To evaluate the efficiency increase due to the similarity detector mechanism, we examine the tradeoff between accuracy and classification when operations/stages are skipped. It takes approximately 10 ms to run the 7-activity GMM classifiers, whereas the similarity measure only requires 0.02 ms to compute. Figure 9 shows the tradeoff between the accuracy and the percentage of GMM classification saved. As the similarity threshold increases more GMM classification operations are skipped and as a result the accuracy of the activity classification is reduced. From the plot it can be observed the accuracy tradeoff ranged from 84% when 4% of the GMM classification is skipped to 60% when 88% is skipped. Even when 73% of computationally expensive GMM operations are skipped, the penalty on the accuracy of the activity classification is only 5%. This is an important result that makes low energy microphone pipelines viable. Note that the classification accuracy drops off quickly when 80% of the operations are saved due to the overly loose similarity threshold.

5.3 GPS Pipeline Results

To evaluate the Jigsaw GPS pipeline, we collect traces of activity inferences and traces of GPS coordinates during weekdays and weekends. The activity inference traces are generated by Jigsaw's accelerometer pipeline. The location traces are recorded using a 1 second GPS sampling interval. We compare the Jigsaw MDP based adaptive GPS duty cycle scheme, an accelerometer augmented GPS duty cycle scheme, and several fix duty cycle schemes using the traces. With the accelerometer augmented scheme, the GPS is sampled every 10 s when there is movement (i.e., the variance of accelerometer samples is over a threshold), and suspend sampling when there is no movement. The sampling intervals of the fix duty cycling schemes are $\{5$ s, 1 min, 2 min, 3 min, 4 min, 5 min, 10 min, 20 min$\}$. The Jigsaw GPS pipeline requires activity inferences as input. Table 10 shows the distribution of activity states in the data set. The most dominant activity is stationary during the weekdays while other activities increase during weekends. Although no user rode a bike during the data collection phase, we still observe some cycling inferences in the trace due to the misclassification of walking or vehicle activities.

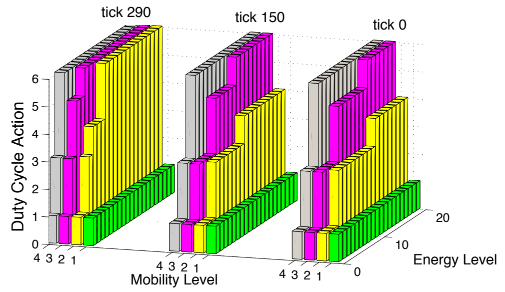


Figure 15: Learned Policy at Different Time Ticks

	stationary	walking	running	cycling	vehicle
weekday	77.3%	10.5%	0.6%	0.4%	11.2%
weekend	38.1%	37.7%	1.8%	0.1%	22.3%

Table 10: Mobility State Distribution in Location Traces

For comparison purposes, the average location errors of all schemes are calculated. Location error is defined as the the mean of euclidian distance between the actual location from the continuous sampling ground truth and the location reported by the different schemes – note, we use the same fixed duty cycles reported in [8]. Intuitively, higher sampling rate schemes consume more energy and have a lower location error and vice versa. The location error and energy efficiency of different fixed duty cycle schemes are shown in Figure 12 and Figure 13 – for the weekday and weekend traces, respectively. In the MDP model, the GPS power budget is set to 25% of N95 phone battery capacity and the sensing duration is set to 10 hours. The total energy budget is discretized into 20 levels and the total duration is discretized into 300 time ticks. As discussed in Section 3.3, the action space contains 6 sensing actions with different sensing intervals, and the state transition matrix of the defined four MDP mobility levels is learned from the motion traces. Figure 15

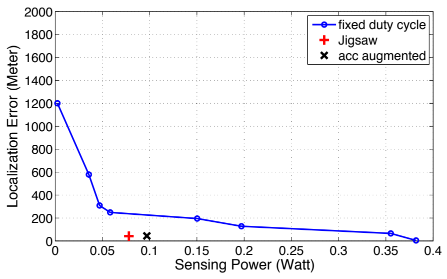


Figure 12: Error vs. Power Tradeoff of Weekday GPS Traces

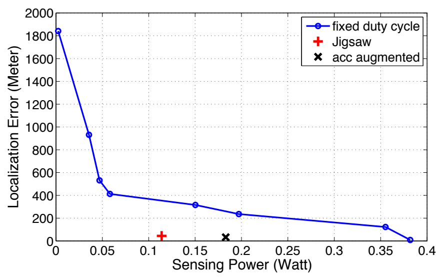


Figure 13: Error vs. Power Tradeoff of Weekend GPS Traces

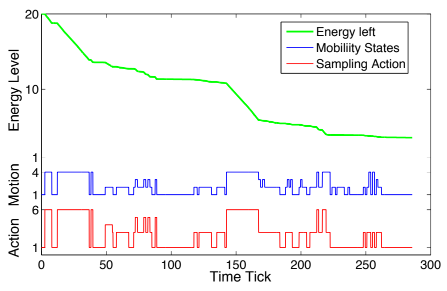
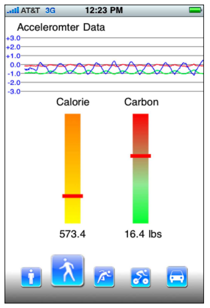


Figure 14: Jigsaw Emulated on a Weekend Trace

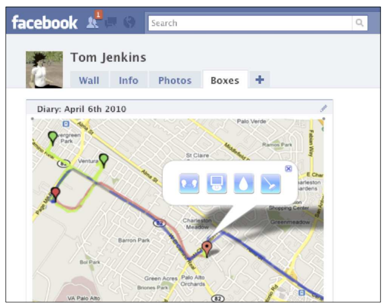
visualizes the learned GPS sampling policy for the weekend traces at three different time ticks (0, 150, 290). The policy changes with the mobility level, energy budget level, and time tick. The mobility level is the dominant factor. When stationary, the system always uses the lowest sampling rate. As the energy budget drains, the system lowers the sampling rate. When the time tick approaches the end of the sensing session, the system tends to choose higher sampling rates to spend the remaining budget. Figure 14 shows the GPS duty cycles for one of the weekend traces. The motion sequence is smoothed by a one minute moving window for clearer visualization. The GPS duty cycle changes along with the user's activity inferences. When the energy budget level is high, the system uses higher sampling rates. As the energy drains, after time tick 50, the sampling policy for walking is lowered. But for the vehicle activity inference with a high associated speed, it still uses a higher sampling rate to keep the location error low. As the energy level falls below energy level 4 after time tick 220, the policy becomes more conservative, reducing the sampling rates even for activity inferences associated with high speeds. Different duty cycle schemes show different tradeoffs between the average localization error and average GPS power consumption, as shown in Figure 12 and Figure 13. *The Jigsaw MDP approach significantly reduces the power usage yet keeps the average error low*. The result of the MDP learned duty cycle scheme is $(41.8m, 0.112W)$, including the power consumed by the accelerometer pipeline. The power consumption is equivalent to a fixed duty cycle using a 4 min sampling interval, while the average location error is between fixed duty cycle schemes sampling at 5s and 1 min. For the weekday traces, the result is $(41.7m, 0.076W)$. Compared to the weekend case, average power consumption is lower due to the larger proportion of stationary activities, while the average error remains almost the same. For the accelerometer augmented GPS sensing scheme, the localization error is only slightly lower than our MDP scheme. However, it does not comply to any predefined energy budget constraint, so it uses more energy in both scenarios, particularly at the weekend when there is more activities.

6 Jigsaw Applications

We implement two simple proof-of-concept applications using the Jigsaw continuous sensing engine. JigMe is an opportunistic sensing application that automatically records a user's daily diary, as shown in Figure 16(b). The GreenSaw application gives users awareness of their daily calorie expenditure and carbon footprint, as shown in Figure 16(a).



(a)



(b)

Figure 16: (a) The GreenSaw application provides carbon footprint and caloric expenditure information, and (b) The JigMe application provides a log of daily activities, significant places, and transportation methods.

GreenSaw encourages environmental and health awareness based solely on the accelerometer pipeline. It provides feedback about health and environmental data computed based on the user's transportation modes and physical activities. The user needs to specify their gender, weight, height, and car model. GreenSaw runs on a jail broken iPhone in order to enable background process support.

In the case of the JigMe application, all three Jigsaw pipelines run continuously in background and produce a time series log of classified activities and location trace. JigMe data is pushed to Facebook and visualized with a map interface, as shown in Figure 16(b). Significant places [4] where the user "sticks" are annotated with tags. Red tags indicate long stays whereas green tags identify short visits. The location trajectory is color coded by accelerometer inferences: green for walking, red for cycling/running, blue for vehicle, and stationary is omitted, such that the transportation methods between places can be clearly identified. When clicking on a tag, a pop up window shows the recognized sound activities in that location. Figure 16(b) shows that four activities are recognized in the user's home: conversation, vacuuming, washing, and typing. The battery life varies due to different phone usage, user behavior and context. The average battery life of the JigMe application on a Nokia N95 is 16 hours with moderate phone use. This compared very well to other continuous sensing applications. For example, the CenceMe [16] applications runs for 6 hours on the same Nokia N95 phone. Recent rule based duty cycled applications EEMSS [29] operates for about 11 hours. JigMe benefits from more sophisticated pipelines which are more resilient to errors where the systems battery performance is not depended on hand-tuned fix duty cycles.

7 Conclusion

Supporting continuous sensing applications on mobile phones is very challenging. The low power design of sensor pipelines is critical to the growing interest and success of sensing applications on mobile phones. In this paper, we present the design, implementation and evaluation of the Jigsaw continuous sensing engine running on the Nokia N95 and the Apple iPhone. Jigsaw is designed to be extensible so as new sensors come online new pipelines can be constructed. Applications can bind these pipelines together as needed and configure them to meet specific needs of applications. Jigsaw performs all the sensing and classification processing exclusively on the mobile phone without undermining the regular usage of the phone. Novel design ideas are introduced in the paper to conserve battery life without sacrificing the accuracy of the sensing system. Specifically, we presented methods that: (i) allows inferences to be robust to different phone hardware, orientation and body positions, (ii) adaptively switches the depth and complexity of sensing process based on the quality of the input data, and (iii) preserves power by taking into account the longer-term mobility and behavior patterns of the user to intelligently trigger power-hungry sensors (e.g., GPS). We believe the flexibility and adaptability of Jigsaw makes it suitable for a wide range of emerging continuous sensing applications for mobile phones.

8 Acknowledgments

This work is supported in part by Nokia, Intel Corp., Microsoft Research, NSF NCS-0631289, and NSF IIS-0845683. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of any funding body. A special thanks goes to our shepherd Jakob Eriksson.

9 References

[1] B. Abdesslem et al. Less is more: Energy-efficient mobile sensing with SenseLess. In *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, pages 61–62. ACM, 2009.

[2] Apple. Introduction to the objective-c 2.0 programming language. Website, 2008.

[3] Apple. iphone sdk. Website, 2008. `http://developer.apple.com/iphone/`.

[4] D. Ashbrook and T. Starner. Using gps to learn significant locations and predict movement across multiple users. *Personal Ubiquitous Comput.*, 7(5):275–286, 2003.

[5] L. Bao and S. S. Intille. Activity recognition from user annotated acceleration data. In *Pervasive*, pages 1–17, 2004.

[6] S. Basu. A linked-HMM model for robust voicing and speech detection. In *Acoustics, Speech, and Signal Processing, 2003(ICASSP'03).*, volume 1, 2003.

[7] S. Consolvo, D. McDonald, T. Toscos, M. Chen, J. Froehlich, B. Harrison, P. Klasnja, A. LaMarca, L. LeGrand, R. Libby, et al. Activity sensing in the wild: a field trial of ubifit garden. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1797–1806. ACM, 2008.

[8] I. Constandache, S. Gaonkar, M. Sayler, R. R. Choudhury, and L. P. Cox. Energy-efficient Localization Via Personal Mobility Profiling. In *The First Annual International Conference on Mobile Computing, Applications, and Services*, 2009.

[9] R. A. Howard. *Dynamic Programming and Markov Processes*. The MIT Press, Cambridge, MA, 1960.

[10] M. B. Kjægaard, J. Langdal, T. Godsk, and T. Toftkjær. Entracked: energy-efficient robust position tracking for mobile devices. In *MobiSys '09: Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 221–234, New York, NY, USA, 2009.

[11] J. Lester, T. Choudhury, and G. Borriello. A practical approach to recognizing physical activities. *Lecture Notes in Computer Science : Pervasive Computing*, pages 1–16, 2006.

[12] D. Li, I. Sethi, N. Dimitrova, and T. McGee. Classification of general audio data for content-based retrieval. *Pattern Recognition Letters*, 22(5):533–544, 2001.

[13] J. Lotters, J. Schipper, P. Veltink, W. Olthuis, and P. Bergveld. Procedure for in-use calibration of triaxial accelerometers in medical applications. *Sensors and Actuators A: Physical*, 68(1-3):221–228, 1998.

[14] H. Lu, W. Pan, N. Lane, T. Choudhury, and A. 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*, pages 165–178. ACM New York, NY, USA, 2009.

[15] M. McKinney and J. Breebaart. Features for audio and music classification. In *Proc. ISMIR*, pages 151–158, 2003.

[16] E. Miluzzo, N. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. Eisenman, X. Zheng, and A. Campbell. Sensing meets mobile social networks: The design, implementation and evaluation of the CenceMe application. In *Proceedings of SenSys08*, pages 337–350. ACM New York, NY, USA, 2008.

[17] D. Mizell. Using gravity to estimate accelerometer orientation. In *ISWC '03: Proceedings of the 7th IEEE International Symposium on Wearable Computers*, page 252, Washington, DC, USA, 2003. IEEE Computer Society.

[18] P. Mohan, V. N. Padmanabhan, and R. Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In T. F. Abdelzaher, M. Martonosi, and A. Wolisz, editors, *SenSys*, pages 323–336. ACM, 2008.

[19] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, R. West, and P. Boda. Peir, the personal environmental impact report, as a platform for participatory sensing systems research. In *Proceedings of Mobisys09*, pages 55–68. ACM New York, NY, USA, 2009.

[20] Nokia. Symbian c. Website, 2009. `http://www.forum.nokia.com/Technology_Topics/Development_Platforms/Symbian_C`.

[21] D. Peebles, H. Lu, N. Lane, T. Choudhury, and A. Campbell. Community-Guided Learning: Exploiting Mobile Sensor Users to Model Human Behavior. In *Proc. of 24th AAAI Conference on Artificial Intelligence*, 2010.

[22] V. Peltonen, J. Tuomi, A. Klapuri, J. Huopaniemi, and T. Sorsa. Computational Auditory Scene Recognition. In *IEEE Intl. Conf. on Acoustics Speech and Signal Processing*, volume 2. IEEE; 1999, 2002.

[23] N. Ravi, N. Dandekar, P. Mysore, and M. L. Littman. Activity recognition from accelerometer data. *IAAI-05*, 2005.

[24] S. Reddy, J. Burke, D. Estrin, M. Hansen, and M. Srivastava. Determining transportation mode on mobile phones. In *Proceedings of The 12th IEEE Int. Symposium on Wearable Computers*, 2008.

[25] E. Scheirer and M. Slaney. Construction and evaluation of a robust multifeature speech/musicdiscriminator. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, volume 2, 1997.

[26] D. Smith, L. Ma, and N. Ryan. Acoustic environment as an indicator of social and physical context. *Personal and Ubiquitous Computing*, 10(4):241–254, 2006.

[27] M. Spina and V. Zue. Automatic transcription of general audio data: preliminary analyses. In *Spoken Language, 1996. ICSLP 96. Proceedings.*,, volume 2, 1996.

[28] Y. Wang, B. Krishnamachari, Q. Zhao, and M. Annavaram. Towards the Tradeoff between Energy Efficiency and User State Estimation Accuracy in Mobile Sensing. In *Proceedings of The First Annual International Conference on Mobile Computing, Applications, and Services*, 2009.

[29] Y. Wang, J. Lin, M. Annavaram, Q. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 179–192. ACM, 2009.

[30] I. Witten. *Weka: Practical Machine Learning Tools and Techniques with Java Implementations*. Dept. of Computer Science, University of Waikato, 1999.

[31] T. Zhang and C. Kuo. Audio-guided audiovisual data segmentation, indexing, and retrieval. In *Proceedings of SPIE*, volume 3656, page 316. SPIE, 1998.

[32] F. Zheng, G. Zhang, and Z. Song. Comparison of Different Implementations of MFCC. *Journal of Computer Science and Technology*, 16(6):582–589, 2001.