

Ada: Context-Sensitive Context-Sensing
on Mobile Devices

by

Yu-Han Chen

B.S., Computer Science and Information Engineering (2010)
National Taiwan University

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

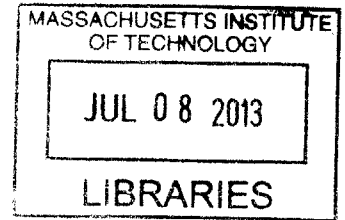
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2013

© Massachusetts Institute of Technology 2013. All rights reserved.

ARCHIVES



Author.....
Department of Electrical Engineering and Computer Science
May 10, 2013

Certified by.....
Hari Balakrishnan
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by.....
Leslie A. Kolodziejski
Chair of the Committee on Graduate Students

**Ada: Context-Sensitive Context-Sensing
on Mobile Devices**

by

Yu-Han Chen

Submitted to the Department of Electrical Engineering and Computer Science
on May 10, 2013,

in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

This thesis describes the design, implementation, and evaluation of Ada, a context-sensing service for mobile devices. Ada explores new points in the accuracy-energy-responsiveness design space for mobile context sensing. The service exports an API that allows a client to express interest in one or more context types (mode-of-movement, indoor/outdoor, and entry/exit to/from named regions), and subscribe to specific modes within each context (e.g., “walking” or “running”, but not any other movement mode). Each context type in Ada can be in one of a set of mutually exclusive states. Each context has a detector that returns its estimate of the mode. To achieve high accuracy and low energy consumption, the detectors take both the existing context and the desired subscriptions into account, adjusting both the types of sensors and the sampling rates. To accurately determine the movement mode, Ada uses a new peak frequency feature from acceleration magnitudes, combining it with two other features. We present results from trace-driven experiments over carefully labeled data from real users, finding that our mode-of-movement detector achieves an accuracy of 93%, out-performing previous proposals like UCLA (55%), EEMSS (83%) and SociableSense (72%), while consuming between 2 and 3× less energy.

Thesis Supervisor: Hari Balakrishnan

Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

First of all, I would like to express my deepest gratitude to my advisor, Hari Balakrishnan, who has provided me with immeasurable opportunities and constant support. I have been greatly inspired by his wisdom, creativity, and vision. Doing research is challenging, but his guidance always prevents me from heading into possible catastrophes, his optimism encourages me to conquer all the difficulties ahead, and his recognition makes every effort worthwhile. Outside of research, he is so down to earth and humorous that you might sometimes forget he is one of the leading computer scientists in the world.

My thanks go to my academic advisor, Daniela Rus; her advice has been a great help in both academic studies and life. I still remember when she shouted “tough girl” at me as I decided to take two TQE classes in my first semester; believe it or not, those two words tremendously boosted my confidence, and helped me get through my hardest times at MIT. Also, I would like to offer my special thanks to Li-Shiuan Peh and Sam Madden, for their warm support, and encouragement in many unexpected ways.

To my undergraduate advisors, Hao-Hua Chu and Polly Huang, they are the reasons that brought me here to MIT. They boldly offered me the opportunity to take part in the PipeProbe project as an undergraduate student, despite the fact that I knew nothing about research. They guided me through each step in completing a paper, and put all of their trust in me to excel. I will always be thankful for their support and recognition.

My thesis work would have never been possible without the help from my amazing collaborators. I am particularly grateful for the assistance given by Anirudh Sivaraman. He contributed generously to this project, helped me design and implement the system, corrected my writing, and taught me all the skills required to be a better coder. Most importantly, he is always willing to spend time chatting with me, sharing his life experiences, and providing advice on how to get the most out of the PhD experience and maintaining a healthy and happy life at the same time. I thank Lenin Ravindranath – for being a key person on this project. His knowledge and inspiration in this field are simply invincible. Without his advice and persistence, this work would not have been so successful. To Somak Das, for assisting me in completing all of the experiments. Working with him has been a lot of fun, and I

will never forget the afternoon we spent together in zero-degree weather attempting to get a GPS fix. To Alvin Cheung – the king of free food – thanks for being the best mentor, guiding me through disappointments, pointing out flaws in my ideas or algorithms, providing me numerous survival tips, and letting me believe everything is possible.

My special thanks are extended to my labmates in the NMS group. Shuo Deng has greatly helped me adapt to the research environment at MIT. She patiently taught me how to use the power monitor (more than two times), graciously listened to my grumbles, and shared many secret discounted deals with me. Katrina LaCurts is the person who brings fun into the lab. Without her, the plants in our office would never have gotten their names, the NMS group would never have had a group outing, and the whiteboard eraser would never have found its wonderful home. She is the one who turns the intense atmosphere at MIT into something a lot more playful and loving. Keith Winstein, one of the brightest men I have ever met in my life, is very selfless. He is always ready to share his wisdom and experiences with others, and he tries hard to help everyone find ways to enjoy their PhD life like he does. I especially thank him for the feedback on my paper drafts, all his comments hit the nail on the head and made me clearly understand what I should do to improve it. I would also like to thank Raluca Ada Popa for giving me the shirt with her superpower, Victor Costan for teaching me how to give an impressive short talk, Jonathan Perry for many fruitful research discussions, and Ravi Netravali for all the fun NBA chats.

I would also like to thank many of my friends who fill my MIT experience with happiness. Their company is a constant reminder that there is much more to life than research. To Ann Lee, for being as such a wonderful friend, putting up with all my bad jokes, and buying me food when I am suffering with a deadline. I enjoy the time we spend together trying to prove a language is NP-complete, attending the most relaxing PE class, exploring new restaurants, and supporting each other through ups and downs. To my roommate, ShinYoung Kang, who has been the greatest victim of this work. Thanks for never being exasperated with my late hours and dirty dishes. And thanks for being like a mom, reminding me to maintain a better work-life balance, educating me about the common sense in everyday life, and listening to my frustration with care and understanding. To Kai-Chien Yang, for the friendship in the past twelve years. She creates a home away from home for me and is always there for a

chit-chat, gossip, and many laughs. Furthermore, I thank Chia-Hsin and Yu-Hsin for helping me pass my TQE, Shu-Heng for organizing all the wonderful gatherings, Yehes and Chern for collecting a huge amount of biking and running data for this project, and Hsin-Jung, Pan, Kevin, Marcus, and Tony for filling my life with many smiles.

Last and most importantly, my deepest thanks to Dad, Mom, Kimberly, and my family, I owe you everything for making me the person I am today. It is your encouragement that motivates me to keep marching ahead and it is your endless support that makes all this possible. I also thank little Senya and Vasya, for bringing tremendous joy into my life. Finally, I would like to thank Erh-Kang Tsao, for always believing in me, giving me all you have, and loving me as the way I am. It is just a miracle to have you in my life.

Contents

1	Introduction	15
2	Related Work	19
2.1	Activity Detection	19
2.1.1	Custom Wearable Sensors	19
2.1.2	Mobile Devices	21
2.2	Energy-Efficient Activity Abstractions	23
2.3	Location-Based Context Detection	24
3	Design Overview	27
3.1	Ada’s Architecture	27
3.2	Power Consumption in Context Sensing	28
4	Context Detectors	33
4.1	Mode-of-Movement Detector	33
4.1.1	Features	34
4.1.2	Modeling Feature Distributions	38
4.1.3	Fusing Sensor Data: Soft Voting	40
4.1.4	Context-Sensitive Sampling to Save Energy	40
4.2	Indoor-Outdoor Detector	41
4.3	Geofence Detector	42
5	Evaluation Method	45
5.1	Trace Collection	45

5.2	Splitting up the Data	46
5.3	Trace-Driven Simulation	46
5.4	Implementation of Related Work	48
5.4.1	UCLA*	48
5.4.2	EEMSS*	48
5.4.3	SociableSense*	49
6	Results	51
6.1	Mode-of-Movement Detector	51
6.1.1	Accuracy and Energy as a Function of Callback Subscription	52
6.1.2	Accuracy and Energy as a Function of Latency	55
6.1.3	Relative Importance of Different Features	55
6.1.4	The Effect of Combining Accelerometer with WiFi/GPS and Smoothing	57
6.2	Indoor-Outdoor Detector	59
6.3	Geofence Detector	60
7	Summary and Future Work	63
7.1	Summary and Contributions	63
7.2	Future Work	65

List of Figures

3-1	Ada’s architecture	27
3-2	Power consumption of different sensors on three phone models (HTC Vivid, HTC Sensation, and Samsung Galaxy Nexus)	29
4-1	Time series of acceleration magnitude	34
4-2	CDF of peakf frequency	35
4-3	Mode-of-movement detector processing algorithm	36
4-4	Estimating user’s speed of movement using the similarity between two WiFi fingerprints under different sampling intervals. When the user is static (driving), the similarity between two consecutive WiFi fingerprints are high (low) across all sampling intervals	37
6-1	Accuracy as a function of subscription. Ada achieves higher accuracy than other schemes always	53
6-2	Energy consumption as a function of subscription. Ada consumes lower energy than other schemes always	54
6-3	Energy and accuracy versus latency. Ada’s mode-of-movement detector trades off latency of detection for decreased energy consumption while maintaing the same accuracy	56
6-4	The effect of combining accelerometer with WiFi/GPS and smoothing	58
6-5	Geofence power percentiles. Ada’s geofence detector consumes low energy and its energy consumption increases as the number of geofences increases	60

List of Tables

1.1	Accuracy and energy consumption of different mode-of-movement schemes. Ada’s mode-of-movement detector outperforms previous proposals like UCLA* [48], EEMSS* [57], and SociableSense* [42], while consuming between 2 and 3x less energy	17
2.1	Features of related systems in activity detection using mobile devices	21
3.1	Ada’s context detectors and the sensors used in each of the detector	28
5.1	The state machine of EEMSS	49
6.1	The features and their corresponding KDE variances (h^2) used in Ada’s mode-of-movement detector. We use cross-validation on the training set to obtain the best set of parameters, and freeze them before we see the testing data	52
6.2	% accuracy with feature subsets, static callback	57
6.3	% accuracy with feature subsets, all 5 callbacks	57
6.4	% accuracy while adding other features	59
6.5	Energy and accuracy for indoor-outdoor detector	59

Chapter 1

Introduction

For about two decades, the mobile computing community has recognized that the ability to infer a mobile user’s context (i.e., properties of the user’s activity and location) is useful in a wide range of mobile applications. Some examples include situated messaging [44], location-aware advertising [6, 19], and lifelogging [21, 27, 10]. Several papers have been published on detecting user activity and location-based attributes (see Chapter 2). Despite this significant work, however, there is no easily usable context-sensing service available on standard smartphone platforms. There are a few reasons for this. First, current context recognition systems are not accurate enough to provide a general context-sensing service; second, most systems are not energy-efficient in how they use the mobile device’s sensors and process sensor data; third, none of the prior work exposes context detection as a programming abstraction. This work describes the design, implementation, and experimental evaluation of Ada, a system that provides an accurate and battery-efficient context-sensing service with a simple API for Android and Windows Phone devices.

Ada provides three context detectors: (1) mode-of-movement, which determines if the user is static, walking, running, biking, or driving; (2) indoor/outdoor, which determines whether the user is inside or outside a building; and (3) geofences, which determines if the user has entered or exited one of a set of named location-based regions. Ada is extensible to allow other detectors, which could reuse the implemented sensor processing and fusion methods.

In Ada, each of these three context types has multiple modes (i.e., multiple possible

states), but the user can be in only one mode at any time. This restriction simplifies the API, and provides opportunities to reduce energy. A client subscribes to one or more contexts and to one or more modes within each context. For example, a background tasking application could subscribe to “walking” or “running” in the movement context, and to “outdoor” in the indoor/outdoor context; the results may be used to log the total time or distance covered by the user in walking or running outdoors.

Ada incorporates the concept of *context-sensitive* context sensing in depth and applies two themes in its design and implementation:

1. Its detectors determine the sensors to use and the sensor sampling rates taking client subscriptions into account. If the client is interested in walking or running, the sensor sampling rates are different from if the client is interested in biking or driving, for example.
2. Its detectors determine the sensors and sampling rates taking the current mode into account. For example, the geofence detector uses its distance from the nearest geofence in deciding which sensors to sample.

A mobile context-sensing service generally has to navigate a design space that has different trade-offs between detection accuracy, energy consumption, and responsiveness. We define the accuracy of a context detector as the fraction of time when it returns the desired mode correctly. The energy consumed is obtained by measuring the power consumption of different sensors at various sensing rates across different phones, and keeping track of the amount of time spent at each sensing rate, and on a variety of smartphones. The responsiveness is a requirement from the client, which tells Ada how often it wants to be notified.

It is difficult to evaluate context-sensing systems in practice. We develop a rigorous and reproducible experimental method, which does not depend on the specific algorithms used in the context detectors. We gather several carefully labeled traces from real users in different contexts, and stitch them together in our evaluation; the method is in contrast to previous work that shows a specific window of data from one context mode or activity and evaluates the fraction of time the detector gets it correct. By contrast, our method provides a stream of sensor readings across different modes or activities evaluates the detector on how often, within a specified response rate, it gets the answer correct. Our evaluation spans over 50

Scheme	Average Accuracy (%)	Average Energy (Joules)
Ada	93.03	3038.96
UCLA*	54.75	14323.57
Ada	94.21	1024.02
EEMSS*	83.19	4642.12
Ada	98.76	3268.92
SociableSense*	72.71	8584.24

Table 1.1: Accuracy and energy consumption of different mode-of-movement schemes. Ada’s mode-of-movement detector outperforms previous proposals like UCLA* [48], EEMSS* [57], and SociableSense* [42], while consuming between 2 and 3x less energy

hours of trace data drawn from ten users. Table 1.1 summarizes our main results for our mode-of-movement detector.

In addition to the idea of context-sensitive sensing, a simple API with mutually-exclusive subscription modes, and a novel experimental evaluation method, this thesis explains why certain features drawn from the sensed data perform better than others. The mode-of-movement detector is of significant interest both because of its many uses and because of much prior work on the problem. We find that three features, which all work over the time-windowed magnitude of the three-axis acceleration sensor data: the mean, standard deviation, and peak frequency (a feature that we propose here), work better than numerous other previously proposed features.

Chapter 2

Related Work

There has been a variety of related work in the area of context-sensing. To illustrate clearly the contribution of Ada and each of Ada’s detector, we divide related work into three main areas: activity detection, energy-efficient activity abstractions, and location-based context detection.

2.1 Activity Detection

Many systems exist to infer human motion activities, movement modes, or postures. Related work in this domain can be grouped based on the types of devices used to implement the system: custom wearable sensors and mobile devices. In this section, we discuss each category in detail with information on how Ada’s mode-of-movement detector builds on or differs from the ones described.

2.1.1 Custom Wearable Sensors

A majority of research has investigated various methods to determine user’s activity using customized wearable sensors. Farrington et al. [15] builds a wearable sensor badge with two two-axis accelerometers to determine whether a person is sitting, standing, lying, walking, or running. To differentiate between sitting, standing, and lying, they examine the raw accelerometer reading of each direction and determine the orientation of the accelerometer. To further distinguish between running and walking, they use the difference between the

maximum and minimum reading of each direction and count the number of times it crosses the average value. However, no experimental results are presented in the paper.

Randell and Muller [43] use a two-axis accelerometer to determine if the user is walking, running, sitting, walking upstairs, downstairs, or standing. The sensor is worn in the trouser pocket with a fixed orientation and sampled at a low frequency (5 Hz) to save energy. From the samples they extract four features including the root mean square and the integration of both axes over the last 2 seconds. They then use a clustering algorithm to infer the user’s activity. With data collected from 10 users, their results shown that they can infer the user’s activity with 85-90% accuracy.

Kern et al. [23], unlike the two previous systems that try to find recurring patterns in the data, this work models the activity explicitly. They mount twelve three-axis accelerometers to all major joints on the human body to capture user’s postures and activities (sitting, standing, walking, upstairs, downstairs, shake hands, write on board, and keyboard typing). Data is sampled at a rate of 92 Hz, and the mean and variance are computed over a window of 50 samples. The paper also investigates the number and placement of sensors and finds out that for “leg-only” activities, such as walking or stairs, sensors on the legs, e.g. hip, are sufficient.

Bao and Intille [8] develop an activity recognition system to identify twenty activities using two-axis accelerometers placed in five locations on the user’s body. This work shows that placing an accelerometer at only two locations (either the hip and wrist or thigh and wrist) did not affect activity recognition accuracy significantly compared to a system with five sensors. Ganti et al. [16], and Gyorbiró et al. [17] also mount multiple accelerometers on different positions on the body to infer a range of activities. Work by [16] further investigates the amount of storage needed without loss of the precision of data collected. They observe that when the user is walking, the number of footsteps taken were between three and six every second. Hence, based on Nyquist’s sampling theorem, they propose that the minimum sampling rate should be at least 25 Hz.

Most work in this domain shares the same goal with Ada – providing better services or applications by being cognizant of user’s movement activities or modes. Unfortunately, wearable devices with a fixed placement are too obtrusive and are inconvenient. Even though

System	Movement Modes	Sensors	
Kwapisz et al. [26]	Walking, Running, Ascending stairs, Descending stairs, Sitting, Standing	Accelerometer	20 Hz
Miluzzo et al. [34]	Sitting, Standing, Walking, Running	Accelerometer	32 Hz
Sohn et al. [50]	Static, Walking, Driving	GSM	1 Hz
Anderson et al. [7]	Static, Walking, Driving	GSM	-
LOCADIO [25]	Static, Moving	WiFi	3.2 Hz
Zheng et al. [58, 59]	Walking, Driving, Taking a bus, Biking	GPS	0.5 Hz
Stenneth et al. [51]	Static, Walking, Biking, Driving, Taking a train	GPS	Every 15 secs
Mun et al. [36]	Static, Walking, Running, Driving	GSM WiFi	0.5 Hz 0.5 Hz
Reddy et al. [48]	Static, Walking, Running, Biking, Driving	Accelerometer GPS	32 Hz 1 Hz
TransitGenie [53]	Static, Walking, Driving	Accelerometer WiFi GPS	Adaptive
SociableSense [42]	Static, Moving	Accelerometer	Adaptive
EEMSS [57]	Static, Walking, Driving	Accelerometer WiFi GPS	Adaptive

Table 2.1: Features of related systems in activity detection using mobile devices

installing multiple sensors on the human body provides more detailed information, it is only practical for specialized applications and services. Ada differs from most prior work in that we use sensors available on a commodity smart phone along with relaxed requirements on how the device should be worn in addition to being orientation, position, and user agnostic.

2.1.2 Mobile Devices

Mobile devices are increasingly capable of rich multimodal sensing. Today, a wealth of sensors including camera, microphone, GPS, accelerometers, proximity sensors, pressure sensors, and light sensors are already standard on mobile devices. As a result, researchers are looking to use mobile device as a platform for activity detection. Table 2.1 summarizes the features of related systems in activity recognition using mobile phones.

Many research projects have focused on using the three-axis accelerometer on mobile devices to infer user’s activity. Kwapisz et al. [26] aims to infer six activities with forty-three features. This work compares the performance of different classification algorithms and discovers that the multi-layer perceptron performs better than the decision tree and the logistic regression. CenceMe [34] infers user’s current activity with fewer features. It finds that the mean, standard deviation, and the number of peaks per unit time are enough to tell between static, walking, and running. Using the decision tree as the classifier, CenceMe achieves 70-80% accuracy. However, both systems assume the phone is in user’s pocket. They

also do not attempt to detect biking or driving.

Some approaches to detecting human movement modes have used radio signals from GSM or WiFi. Sohn et al. [50] uses the rate of change in GSM cell tower observations to infer whether a user is stationary, walking, or driving. The inference is then used to approximate the user’s daily step count. In [7], similar experiments are carried out by further exploiting the knowledge of human normal behavior with a Hidden Markov Model (HMM). It also investigates the impact of environment (metro versus suburban) on the detection performance and discovers that the system achieves better performance in a suburban environment because people tend to drive at a faster speed than in the metro environment. Even though scanning GSM fingerprints is energy-efficient, GSM fingerprints aren’t universally available across all phones. Some phone models limit the cell information to only the connected tower or do not make it available at all. Moreover, GSM based activity detection systems are unlikely to be able to properly distinguish between activities with similar speed. WiFi based systems, like LOCADIO [25], measure the variation of the signal strength of access points to determine whether or not the user is in motion.

Several systems rely on GPS for sensing movement modes. For instance, Zheng et al. [58, 59] combine GPS with a post-processing step that uses likely transportation modes from a corpus of contributed data for classification. Alternatively, Stenneth et al. [51] combines GPS with GIS information, such as the real-time bus locations, spatial rail and spatial bus stop information. They find that speed is the most effective feature from GPS to infer movement modes. Liao et al. [28] builds personalized models based on learning destinations and routes from historical data to build transportation mode classifiers. Unfortunately, GPS based systems consume significant amount of energy and do not work when the device is indoors. Also, GPS-only solutions perform poorly when classifying of movement modes with similar speed, such as running, biking, and slow driving.

Other work integrates multiple sensors on the mobile device to determine a user’s activity. Mun et al. [36] uses the connected GSM cell tower information with the additional of features derived from WiFi observations for classification. TransitGenie [53] uses data from the accelerometer, WiFi, and GPS to distinguish walking from driving, and uses the prediction to estimate the arrival time of buses. To determine which sensors to use and their sampling

rates, it maintains a finite state machine and only turns on power-hungry sensors only if it is needed. Ada also adopts this notion, but is able to detect more mobility modes.

Closest to our work is Reddy et al.’s work from UCLA [48] (which we will refer to as UCLA* in this thesis), which uses GPS and accelerometer readings for transportation mode inference. This work includes a comprehensive evaluation of the energy and accuracy of the system under various scenarios. We find that Ada outperforms UCLA* in our evaluation on both accuracy and energy. Ada also combines the information from multiple sensors. However, unlike previous work, it adapts the choice of sensors and their sampling rates based on feedback from the detector. We observe that acceleration data, by itself, is good to capture movements, but distinguishing biking from walking and driving is a vexing problem. Augmenting the acceleration-derived features with approximate speed estimates could help this distinction. Therefore, Ada relies primarily on acceleration data, uses WiFi to approximate speed, monitors the density of WiFi APs, and only turns on GPS when WiFi APs are not visible.

2.2 Energy-Efficient Activity Abstractions

Several research projects aim to build an energy-efficient activity recognition system through intrusive instrumentation of the user or a user’s surroundings using sensor boards [56, 52, 31, 22], on-body accelerometers [9, 33, 45], or RFIDs [41]. Instead, our system focuses on activity recognition using commodity smart phones. Systems such as SociableSense [42], Kobe [11], and EEMSS [57] explore energy-accuracy-latency tradeoffs in sensing applications running on smart phones. SociableSense provides an adaptive sampling mechanism using a learning technique based on the theory of learning automata to control the sampling rate of the accelerometer. It maintains a probability of sensing to determine the sampling duty cycle of the accelerometer. The sensing probability is then updated based on the result of sensing action. EEMSS adopts the strategy of hierarchical sensor management to recognize user states as well as to detect state transitions. Its goal is to power only a minimum set of sensors and use appropriate sensor duty cycles to improve device battery life. Ada is motivated by the same end goal of energy-efficiency, but, our approach is different. We

compare Ada to both SociableSense and EEMSS (which we will refer to as SociableSense* and EEMSS* in this thesis) and find that we outperform both systems on both energy and accuracy.

Google [1] recently launched an activity detector API for Android. Its emergence reveals the desperate need for exposing context detection as a programming abstraction. The service detects if the user is currently on foot, in a car, on a bicycle, or static. The service allows applications to request periodic activity recognition updates. For each update interval, the service sends a list of possible activities and the probability of each one to the application. The activities are detected by periodically waking up the device and reading short bursts of low power sensor data to save energy. We are planning to compare this system with Ada’s mode-of-movement detector.

SeeMon [22] achieves energy-efficiency in the context of multiple external sensors attached to a computation device; they do not evaluate their system on mobile phones. ACE [37] uses a different approach, by detecting low energy proxy activities that correlate with the activity that needs to be detected. Ada’s approach is complementary and can be applied to the proxy detectors in ACE. Our approach of context-sensitive sensing may be viewed as a significant generalization of triggered sensing [35].

2.3 Location-Based Context Detection

Several systems reduce the energy cost of localization using a variety of techniques. Some systems [60, 32] adapt their sampling rate to reduce energy cost. A-Loc [29] uses the observation that the required location accuracy varies with location. Where possible, RAPS [38] uses historical data along with the accelerometer in lieu of GPS. CAPS [39] and CTrack [55] sequence cellular base stations to retrieve approximate position. Cleo [30] offloads all GPS signal processing to the Cloud to reduce energy consumption, but requires a modified GPS board to support offloading, increases wireless energy consumption a little (and adds latency). EnLoc [12] allows the client to pick an energy budget which it then uses to achieve the best accuracy subject to the energy budget constraint. Ada’s geofence detector can use these schemes as a blackbox, by deciding when to query them for location, without worrying about

how they work internally. SensLoc [24] shares Ada's goal of detecting named locations, but works best only when the user visits the same location repeatedly.

Chapter 3

Design Overview

In this chapter, we describe the architecture of Ada, the services it provides, and the energy profile of various sensors that guides the design of the system.

3.1 Ada's Architecture

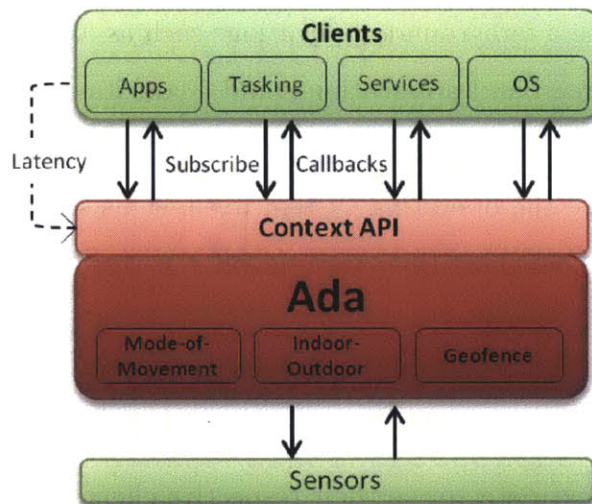


Figure 3-1: Ada's architecture

Figure 3-1 shows the design of Ada and Table 3.1 shows the sensors used in Ada's detectors. A *client* subscribes to one of several available callbacks from any of Ada's context detectors. For instance, a client using movement hints to enhance the performance of WiFi (e.g. [46]) might only be interested in monitoring if the user is static or not. Alternatively, a

Detector	Modes	Sensors
Mode-of-movement	Static, Walking, Running, Biking, Driving	Accelerometer, WiFi, GPS
Indoor-Outdoor	Indoor, Outdoor	WiFi
Geofence	Set of geofences	GPS, WiFi, GSM, Accelerometer

Table 3.1: Ada’s context detectors and the sensors used in each of the detector

fitness application calculating a user’s daily calorie expenditure might be interested in all five callbacks from the mode-of-movement detector. Clients also specify the latency of detection, which specifies how often Ada returns a callback to the client. For example, a reminder application might need to notify the user within a few seconds while a trajectory logging application can tolerate a longer detection latency before it starts logging samples.

We intentionally use the term “client” rather than “application” or “process” because Ada’s APIs support several different use cases. For example:

1. The client could be tasking applications or services [47] that run in the background. These applications run continuously in the background, triggering actions based on the activities. For instance, the tasking application might want to log certain sensory data whenever the user starts running.
2. The client could be a foreground applications such as a web browser or chat plugin. In such cases, the foreground application could adapt its user interface to the activity of the user.
3. The client could be the operating system itself. This scenario is useful when we want to change system behavior (like turning off the screen or setting “airplane” mode) automatically. Ada can also help wireless protocols by providing contextual hints [46].

Ada maintains a list of clients that have subscribed to each of the modes that Ada’s detectors provide. It runs only one instance of a specific detector for each phone and conveys callbacks from the context detector to their respective clients.

3.2 Power Consumption in Context Sensing

To understand the power consumption of sensors on a phone, we performed experiments to quantify the power consumption of GPS, WiFi, GSM, and the accelerometer on three phone models (Samsung Galaxy Nexus, HTC Vivid, and HTC Sensation). We wrote an

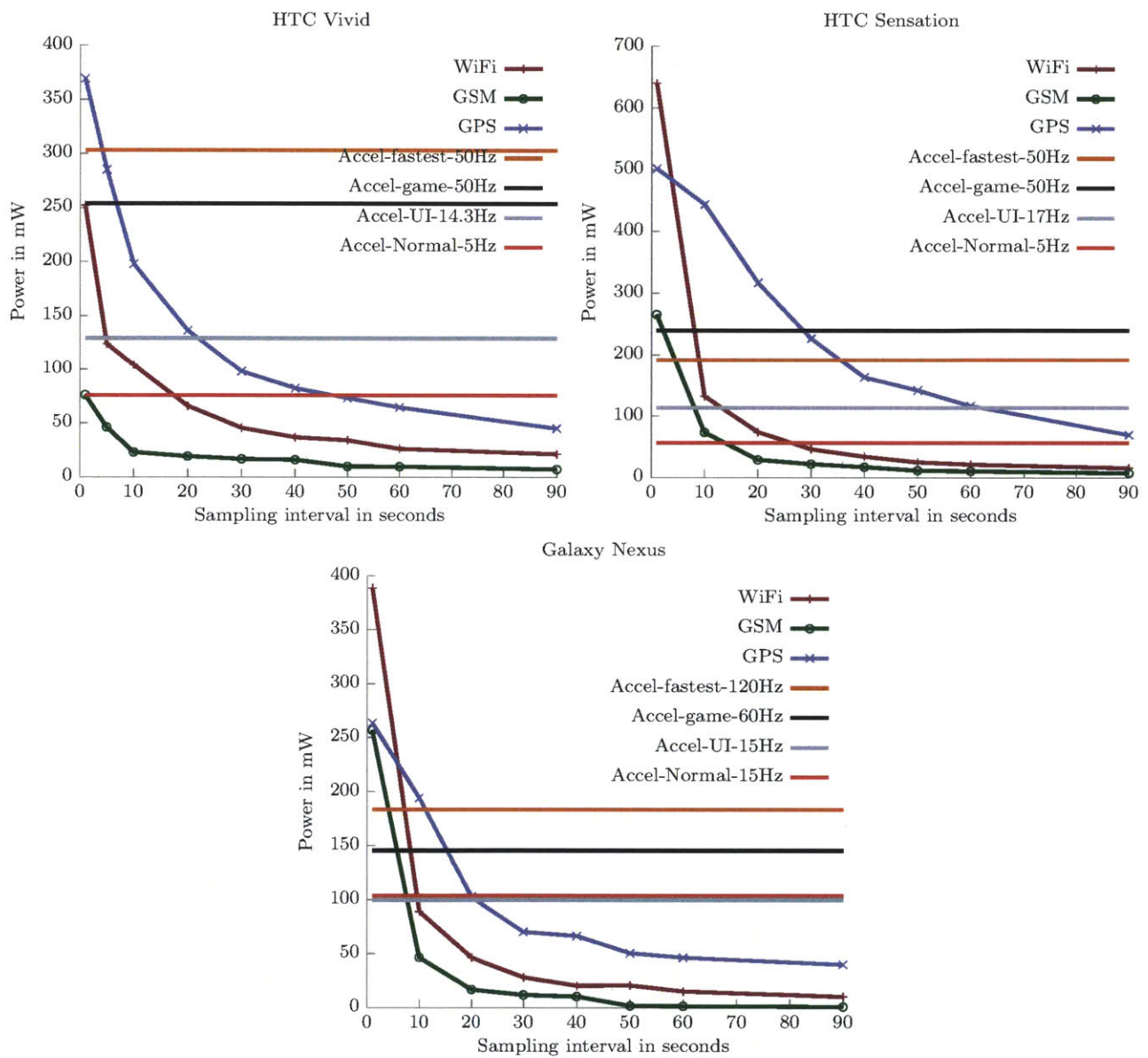


Figure 3-2: Power consumption of different sensors on three phone models (HTC Vivid, HTC Sensation, and Samsung Galaxy Nexus)

Android application to continuously sample the sensor at a given sampling interval. For the accelerometer, we set the mode to each of FASTEST, GAME, UI, or NORMAL. For WiFi and GSM, we used Android’s AlarmManager to schedule a periodic WiFi scan at the specified sampling interval. When the alarm fires, the Android application asks for the names and RSSIs of neighboring WiFi APs (access points) or cell towers. For GPS, we used the Android LocationManager API to specify a sampling interval. We conducted GPS measurements in an open space with good GPS reception to ensure the GPS would acquire a lock.

To accurately measure power consumption of the sensors alone, we halted all running applications, turned off the screen, used the Monsoon Power Monitor to deliver a constant voltage of 3.7V, and measured the long-term average power consumption. The power profiles for three phone models are shown in Figure 3-2.

We make three observations. First, there is a clear difference in the power consumption of different sensors. For instance, on the HTC Vivid, at a sampling interval of 5 seconds, GSM is more than $6\times$ as energy-efficient as GPS.

Second, depending on the sampling interval, WiFi scans on the Sensation, for instance, can consume anywhere between 16 mW and 640 mW.

Third, the accelerometer power consumption increases with sampling rate despite the fact that the accelerometer is itself a low-power sensor. The reason is that Android currently retrieves accelerometer samples by polling the accelerometer at the specified sampling rate. So, the power consumption measured reflects the power consumption of a CPU polling a peripheral for I/O at the specified sampling rate. This could be easily fixed if the accelerometer’s samples were batched and transferred to main memory every second using Direct Memory Access (DMA).

These observations guide our context-sensing methods. The first observation shows that picking sensors based on the specific subscriptions will save energy. The second observation shows that being selective about the sampling interval for a particular sensor is useful. Finally, because the power consumption of the accelerometer goes up with the sampling rate, we do not use a sampling rate higher than 40 Hz.

The measurements above ignore the cost of computation. Any context-sensing algorithm will involve some computation in addition to the sensing itself. For instance, it is common

to window the data to compute specific statistics (such as mean, and variance). Normally, the windowing interval is on the order of a few seconds. The interval between successive predictions from a context sensing schemes is on the order of tens of seconds. At these time scales, computation energy is insignificant compared to sensing energy.

Chapter 4

Context Detectors

In this chapter, we describe the algorithm of the mode-of-movement detector, the indoor-outdoor detector, and the geofence detector. We also illustrate the observations that guide the design of the techniques presented in this chapter.

4.1 Mode-of-Movement Detector

This detector determines which one of five movement modes—static (S), walking (W), running (R), biking (B), or driving (D)—a mobile device (user) is in. A client running on the mobile device may subscribe to any subset of these modes, specifying a response interval, L ; in return, the detector returns its best estimate of the movement mode every L seconds. The movement-mode detector uses data from the three-axis accelerometer, and if available, augmented with data from WiFi position sensors or from GPS. All these sensors are sampled with energy-efficiency in mind, so the amount of data from GPS will be small or non-existent in many cases.

One prior approach to detecting the movement mode (also called “activity” in some earlier work) is to use GPS velocity [48, 57]. Unfortunately, this approach is problematic even when battery life is not a concern because GPS is unavailable indoors, and because vehicles may move at biking or even slower speeds in heavy traffic. Ada supports both indoor and outdoor uses, relying primarily on acceleration data. It uses WiFi to approximate “speed” and GPS when WiFi APs are not visible.

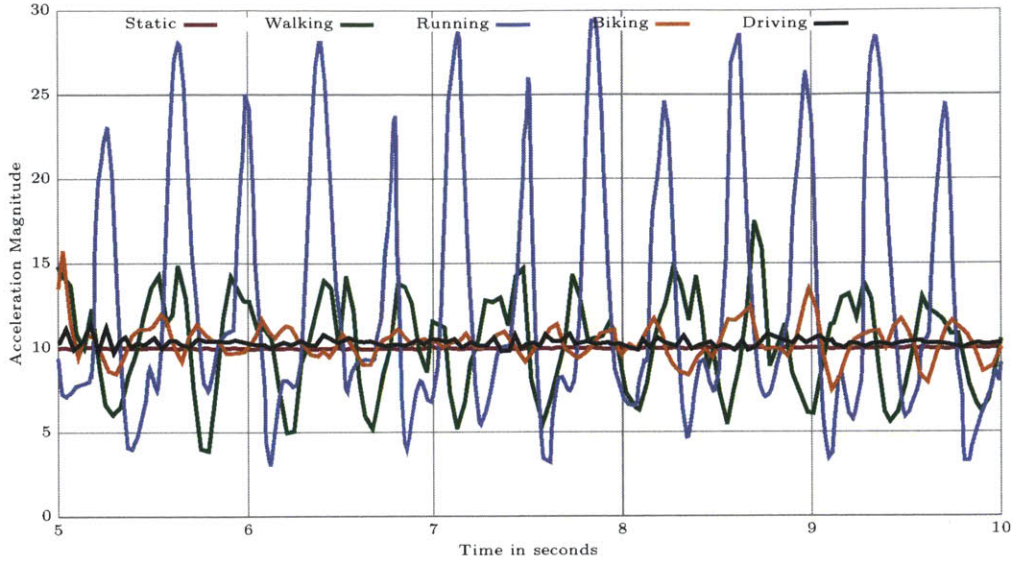


Figure 4-1: Time series of acceleration magnitude

The main contributions of our movement detector over prior schemes are: (1) the use of a new *peak frequency* feature from the acceleration data, used in conjunction with two other known features, (2) a processing algorithm (summarized in Figure 4-3) that combines the different features of a feature vector from one sensor using the Naive Bayes classifier, and fuses the outputs of different sensors using *soft voting*, and (3) an adaptive sensor sampling method that takes both the subscribed callbacks and current mode into consideration to reduce energy consumption. The rest of this section describes these ideas.

4.1.1 Features

The detection problem is a multi-label classification with the labels being S, W, R, B, and D, the five modes mentioned above. For data collected from each sensor, the detector computes feature vectors over which the training phase and online learning method can work.

Accelerometer. As in prior work, we compute the magnitude, $\sqrt{a_{x_i}^2 + a_{y_i}^2 + a_{z_i}^2}$, of each three-axis accelerometer reading. The intuition for this calculation is shown in Figure 4-1, which shows the time series of the magnitude for different movement modes. One can see that many of the modes of interest “look different” on this chart. For example, running has the highest standard deviation when computed over a time window of a few seconds, so

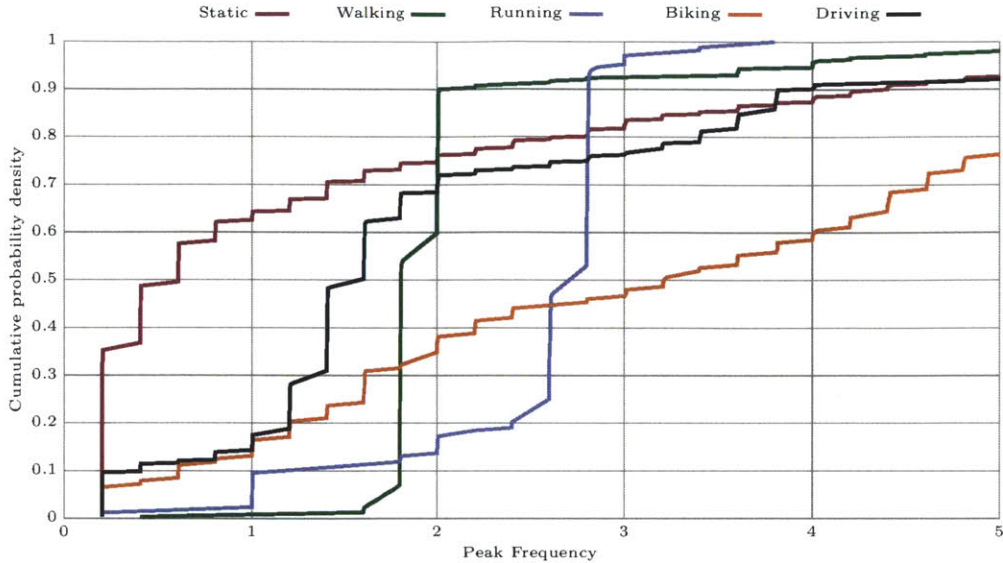


Figure 4-2: CDF of peak frequency

it might be possible to use the windowed standard deviation as a useful feature. So does walking, but people generally walk slower than they run, so one might be able to distinguish between those two modes by looking at the frequency-domain version of the same data and seeing if they peak at different frequencies. Plotting the frequency spectrum does show a spike around 1 Hz to 2 Hz, whereas biking and driving are more evenly spread over a wider frequency range.

This intuition suggests that we want a feature to capture the periodic behavior, another to capture the overall strength of the acceleration (while static, gravity is the only force acting, while other modes will have differing magnitudes), and another to capture the variation of these magnitudes. There are numerous possible features one could construct and try out. We experimented with many possibilities, inspired by previous work on activity detection as well as signal processing methods used in medicine. These features include:

1. Mean acceleration magnitude, μ , calculated over non-overlapping windows of $T = 5$ seconds, previously investigated by [32].
2. Standard deviation, σ , of accelerometer magnitudes calculated over the same windows [53, 48, 42, 57].
3. The peak non-zero (non-DC) frequency, p_f , of the acceleration magnitudes computed over the same time window using the FFT.

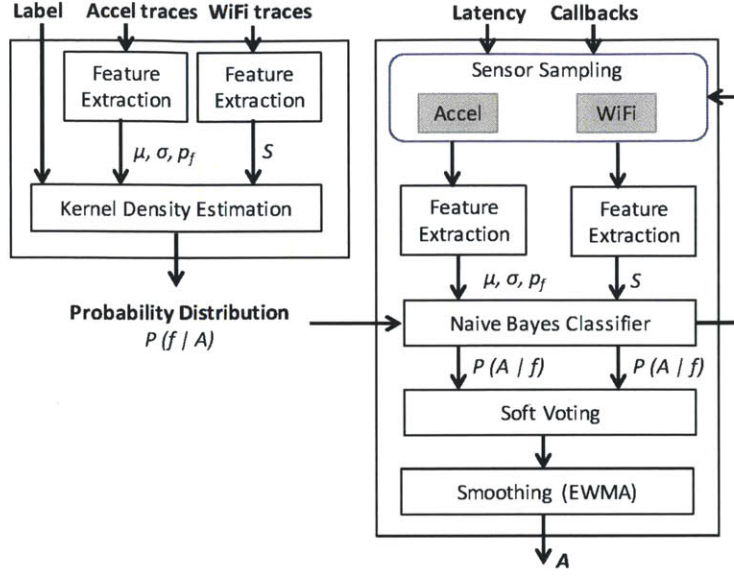


Figure 4-3: Mode-of-movement detector processing algorithm

4. The spectral coefficient of the magnitude of the three-axis acceleration data samples at 1 Hz, 2 Hz, and 3 Hz [48].
5. The spectral entropy of the DFT of accelerometer magnitudes over a time window of $T = 5$ seconds [32].
6. The peak power ratio of the DFT of the accelerometer magnitudes computed over a window of $T = 5$ seconds [53].
7. The line length (or curve length), defined as the sum of the absolute value of the difference between two successive acceleration magnitudes, computed over non-overlapping windows of $T = 5$ seconds. This feature has been used successfully in seizure detection [14].
8. The average nonlinear energy, defined as the average value of $x_i^2 - x_{i-1}x_{i+1}$, computed over non-overlapping windows of $T = 5$ seconds [20].

We conducted controlled experiments with various combinations of the above features, and empirically determined that **the first three features above** perform the best. Our detector therefore processes the timestamped acceleration data to produce a sequence of μ, σ, p_f values, each value calculated over a T -second non-overlapping window. To our knowledge, the peak non-DC frequency (p_f) is a novel feature, and captures the key idea that different movement

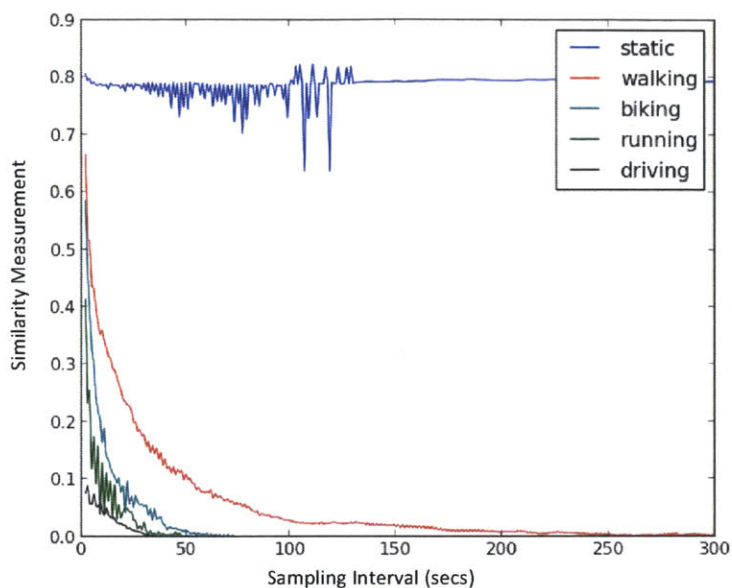


Figure 4-4: Estimating user’s speed of movement using the similarity between two WiFi fingerprints under different sampling intervals. When the user is static (driving), the similarity between two consecutive WiFi fingerprints are high (low) across all sampling intervals

modes have different principal frequencies (see Figure 4-2, which plots the CDF of peak frequency for different modes; the sharp rise in CDF is visible for walking and running, and to a slightly lower degree for driving, compared to the other modes). We don’t need to know what those frequencies are, and indeed they will change between users, but we rely on the reasonable assumption that they are different for any given user. We compare the features we use with some of the alternatives we rejected in §6.1.3. Note that these features are computed over 5-second windows; a smaller window runs the danger that we might not have enough samples to capture key features of the movement mode. As a result, this detector should be used only with L , the response interval, larger than T .

WiFi. Acceleration data, by itself, is good to capture movements, but distinguishing biking from walking and driving is a vexing problem. Augmenting the acceleration-derived features with approximate speed estimates could help this distinction. Because a WiFi scan consumes much less energy than a GPS fix, we use periodic WiFi scans. The detector uses the similarity of observed WiFi access points (APs) and their RSSI values over successive WiFi scans to obtain a crude proxy for the user’s speed of movement.

Each WiFi scan returns a “fingerprint”, defined as a set of (APID, RSSI) tuples. The rate of change of fingerprints tells us something about the rate of movement of the user. To use this idea, we need a measure of the similarity between two WiFi fingerprints. We treat each fingerprint as a vector in the space constructed by all the WiFi APs, and the RSSIs determines its direction in the space. We measure the similarity, S , between two WiFi fingerprints as follows:

$$S(\vec{f}_1, \vec{f}_2) = \frac{\vec{f}_1 \cdot \vec{f}_2}{\|\vec{f}_1\|^2 + \|\vec{f}_2\|^2 - \vec{f}_1 \cdot \vec{f}_2},$$

where \vec{f}_i is a fingerprint scanned at t_i . $0 \leq S \leq 1$, with a larger value suggestive of slower movement. As an example, consider the fingerprints f_1 at time t_1 equal to $\{(ID=1, RSSI=3), (ID=2, RSSI=5)\}$, and f_2 at time t_2 equal to $\{(ID=1, RSSI=2), (ID=3, RSSI=1)\}$. They are converted to $(3,5,0)$ and $(2,0,1)$ in the WiFi AP space, and $S = \frac{(3*2+5*0+0*1)}{34+5-6} = \frac{6}{33}$. We considered other similarity metrics, including one from [55]; we found our metric performed better. The reason is that it more naturally handles fingerprints with partially overlapping or non-matching APs.

Figure 4-4 shows the similarity value between two consecutive WiFi fingerprints under different sampling intervals. We randomly choose one trace for each activity and take the average of all the similarity values with the same sampling interval. As shown in the figure, static has the highest similarity value across all sampling intervals, and the value does not vary with different sampling intervals. As for driving, it has the lowest similarity value and the value decreases quickly when the sampling interval increases. Biking has a lower similarity value than walking. By incorporating the speed estimation, it helps us further distinguish between biking from walking and driving.

GPS. Finally, we use GPS velocity readings as a feature, but only when the WiFi AP density falls below a threshold.

4.1.2 Modeling Feature Distributions

Armed with these features from each sensor, the next step is to train the algorithm. For each sensor, we use labeled training data to produce a conditional probability distribution, $\mathbb{P}(\vec{f}|A_i)$, where \vec{f} from the acceleration data has three components, μ, σ, p_f , and for the other

sensors has only one component, and A_i is a movement mode. We use the standard kernel density estimation (KDE) [40, 49] technique to model the feature distributions. KDE has a number of good properties for our purposes. First, it does not assume a single Gaussian, and can handle multimodal distributions well. Second, the tunable bandwidth parameter in KDE compensates for the difference between feature distributions across users. Third, the model is easy to update. We can incorporate new training samples by appending them into the current training set and simply extending the summation that computes the probability density.

We model the probability density of a feature, conditioned on a specific mode, as a sum of normalized Gaussians. Each Gaussian has the same variance h^2 and has mean equal to one of the training data samples. The parameter h is tunable to trade off between overly smoothing the data (large h) and overfitting the data (small h). We explain how we pick h in §6.1 using cross-validation on the training data set.

Our goal, of course, is to determine $\mathbb{P}(A_i|\vec{f})$, for which we apply Bayes’ rule:

$$\mathbb{P}(A_i|\vec{f}) = \frac{\mathbb{P}(A_i \cap \vec{f})}{\mathbb{P}(\vec{f})} = \frac{\mathbb{P}(\vec{f}|A_i) * \mathbb{P}(A_i)}{\mathbb{P}(\vec{f})}.$$

We assume that the prior distribution $\mathbb{P}(A_i)$ is known; in our implementation, we set them all to be equal. The denominator is a normalization factor that does not need to be separately determined.

Decomposing and recombining sensor features with Naive Bayes. The acceleration-derived feature vector has three components. A multi-variate conditional distribution $\mathbb{P}(\vec{f}|A_i)$ is unwieldy. We use a standard Naive Bayes classifier, which separates the components, and then multiplies the conditional probabilities of the different components together. However, the result may produce a zero posterior probability. We need to “regularize” the result, for which we apply a standard m -estimator [13]: given a probability mass function $\mathbb{P}(A)$, the probability estimate $\mathbb{P}(\hat{A})$ using the m -estimator is

$$\mathbb{P}(\hat{A}) = \frac{N\mathbb{P}(A) + \phi\mathbb{P}(\bar{A})}{N + \phi},$$

where $\mathbb{P}(\overline{A})$ is a prior probability and ϕ determines how much weight we attribute to the prior $\mathbb{P}(\overline{A})$. We use the uniform prior, and $\phi = N/5$ because we have five modes in all.

4.1.3 Fusing Sensor Data: Soft Voting

How should the detector combine the posterior probabilities $\mathbb{P}(A_i|\vec{f})$ obtained from the different sensors (specifically, accelerometer and WiFi or accelerometer and GPS)? One approach is to consider the WiFi or GPS feature as another element in a bigger feature vector and apply the technique described above over a four-dimensional vector. The problem with this approach is that it would weigh the acceleration data considerably higher than the WiFi or position data, and not allow us to explicitly control the weight given to these two independent sensors. Moreover, it would not function well when one of the sensors did not provide data, or provided data at a different rate. With soft voting, we take the product of the posterior probabilities for each movement mode (each mode has a “vote” given by this product). If we want to treat one sensor as more important than another, we can associate a higher multiple with the posterior probability computed from that sensor’s features.

Soft voting helps disambiguate some tricky cases. For example, if the user is biking, the acceleration computations alone might confuse biking with walking or running (but not driving), giving walking (or running) and biking comparable probabilities. On the other hand, WiFi or GPS might confuse biking with driving because they may have similar speeds, but not with running or walking. After the “soft vote” using the product, we are far more likely to infer the correct movement mode.

The final step in the algorithm smooths these vote results using an exponentially weighted moving average (EWMA) filter. The detector returns the movement mode with the highest value of the smoothed soft vote.

4.1.4 Context-Sensitive Sampling to Save Energy

To make the mode-of-movement detector energy-efficient, we adapt the choice of sensors and their sampling rates based on feedback from the detector, the client-specified subscription, and the response interval L . We discussed how the acceleration data performs well in de-

tecting that the activity is static, walking, or running, and incorporating WiFi or GPS gives only marginal gains for these modes. Therefore, we use only the acceleration data when the client subscribes to any subset of these three modes, and turn off WiFi and GPS sensing.

In other cases, we average the accelerometer feature vectors every L seconds and scan the WiFi or GPS sensor every L seconds to conform to the required response latency. We continuously monitor the WiFi AP density and only turn on the GPS when the WiFi AP density falls below a threshold. We also adapt the sampling rate of accelerometer using the posterior probability $\mathbb{P}(A_i|\vec{f})$. The detector starts sampling the accelerometer at the lowest sampling rate, and then ramps up to the next higher rate (accelerometer rates are discrete) every time the posterior of any of the subscribed modes crosses 0.2 and ramps down when the posterior of any of the subscribed modes crosses 0.8.

4.2 Indoor-Outdoor Detector

For the indoor-outdoor detector, the set of mutually exclusive states is indoor and outdoor. To tell if a user is indoors or outdoors, we use variation in WiFi RSSI indoors and outdoors. Specifically, we employ two features of WiFi scans. First, the fraction of good AP sightings in a *micro window* μ . We define a sighting to be “good” if its RSSI is greater than a per-phone threshold (set to 80 for the Galaxy Nexus)¹ Second, the average RSSI across all AP sightings in the same micro window μ .

To train, we sample WiFi scans at 1 Hz and set μ to 5 seconds. Conditioned on the user’s state(indoor or outdoor), we fit a Gaussian distribution to each of the two features listed above.

During classification, we have four time scales of interest:

1. The developer specified latency L .
2. A sliding *macro window* M which is set to L .
3. A sliding *micro window* μ over which the two features (fraction of good APs and average RSSI) are computed. μ is set to $\frac{L}{2}$.
4. The WiFi scan interval I which is set to $L/4$.

¹Per-phone calibration is required to determine a “good” threshold, but need not be repeated for every user using that type of phone.

To classify, we first compute both feature vectors over the micro window. Since the micro window is twice as large as the scan interval, we are likely to include at least two scans while computing both features. Next, we average the feature vectors from all micro windows belonging to the same macro window to generate a prediction feature vector \vec{F} . Since the macro window is twice as large as the micro window, we typically average at least two feature vectors while computing \vec{F} . Lastly, we use the prediction feature vector to compute the likelihood that \vec{F} was observed indoor or outdoor. To compute the likelihood of \vec{F} , i.e., $P(\vec{F}|A)$ (where A is either indoor or outdoor), we use the Naive Bayes classifier.

4.3 Geofence Detector

Geofencing is a way of marking a virtual boundary around a geographical region. Geofences are typically defined as circular regions using a center (expressed in latitude and longitude) and a radius. A Geofence detector monitors the location of the user and detects if the user has entered a geofence of interest. As an example, an app could let a user create a geofence on a map and trigger a notification when she enters it. Many tasking applications [3, 5, 4] including iOS location-based reminders [2] have been built around Geofencing.

Naive implementations of a geofence detector can either be energy-intensive or have poor accuracy. For instance, constantly sampling the GPS is not energy-efficient despite often being accurate. At the other extreme, solely using a low-power position sensor might create a lot of false positives and false negatives.

Ada includes a geofencing detector that is both accurate and energy-efficient. The detector uses GPS, Wifi and cellular radios to get location samples. The geofencing detector adapts both the sampling interval and the location sensor to the current context of the user. If the user is far from the nearest geofence, the detector uses a low-power sensor and samples at a larger interval. As the user nears a geofence, it switches to a high power sensor and samples more frequently to check if the user have entered the geofence.

As input, our geofence detector takes a list of geofences that the client has subscribed to and a latency parameter (l) similar to the mobility detector. The algorithm varies two parameters based on the current context of the user: when to sample next (t), and which

sensor to use on the next sampling instant (s). We set t equal to $\frac{minD}{maxSpeed}$ where $minD$ is the minimum distance between the current location and the nearest geofence accounting for the accuracy in the location sample (for e.g. GSM locations can have errors up to 2 miles). If the calculated t is less than l , we reset t to l (i.e. the latency of detection the client is willing to tolerate). We set a lower bound of 1 hour on the sampling interval t as a failsafe. We choose $maxSpeed$ to be 150 miles/hour under the realistic assumption that the user is unlikely to ever travel faster than that speed on roads (true of the US). To set s , we look at the current value of $minD$. If $minD$ is larger than the error bound of GSM (we use 2 miles as the GSM error [55]), we choose GSM. Alternatively, if $minD$ falls between the error bounds of WiFi (we use 200 meters as the Wifi error [54]) and GSM, we sample WiFi. Lastly, if $minD$ is less than the error bound of WiFi, we switch to GPS. As soon as we find the user inside a geofence, we trigger a callback for that particular geofence.

When the user is inside a geofence, we use a similar technique to adapt the sampling time and the sensor used (based on the size of the geofence and the distance to the next nearest geofence). As an optimization, we also use the mobility detector to detect if the user is static or moving and use it to refine the next sampling time.

Chapter 5

Evaluation Method

We evaluate all three Ada detectors using trace-driven simulation. We describe our evaluation in four parts: §5.1 focuses on trace collection, §5.2 describes how we split up the data into training and testing sets, §5.3 explains how we use these in our simulator for testing, and §5.4 describes our implementation of three related work for Ada’s mode-of-movement detector to compare against.

5.1 Trace Collection

We developed an Android application that collects traces of users engaged in different activities (which correspond to distinct modes of each context). The application collects sensor data by sampling the GPS at 1 Hz and accelerometer at its maximum sampling rate, and by scanning WiFi every second. To collect ground truth, the user tags her current activity on a UI. To avoid errors, users record traces of specific activities rather than tagging their activities over the course of a day. Hence, any recorded trace will have exactly one activity throughout the trace. This approach does suffer from the drawback that no data is collected during activity transitions, but provides us clean, labeled data for each activity. These labeled data-sets can then be “stitched together” to emulate activity transitions.

Our mode-of-movement dataset spans 50 hours and is collected from ten users in an urban area. However, we were unable to get trace data for all activities from a single user. Therefore, we create four “virtual” users, called “Galaxy”, “Sensation-1”, “Sensation-2”,

and “Vivid”, ensuring that, for each virtual user, for any activity, the data is collected from exactly one real user. We use virtual users in the experimental results reported in §6.

5.2 Splitting up the Data

We split all our collected data, for each phone model, into 4 roughly equal folds. We try and ensure two properties while dividing the data. First, each fold has roughly equal durations of every activity. Second, in every fold, for any activity, all data for that activity is collected from exactly one user. To train our detectors, we extract feature vectors relevant to the context detector from all the traces in the given set of folds. Specifically, we generate a set of (feature-vector,ground-truth) tuples from the training set that can be directly used to train our detectors. To test the detector, we pick another set of folds, create *snippets* of each activity, and then stitch these traces together into a longer one for simulation.

5.3 Trace-Driven Simulation

We use the traces collected earlier to run a trace driven simulation. Before traces are input into the simulator, they are sanitized to create snippets. A *snippet* is a correctly labeled, timestamped sequence of sensor data corresponding to a single mode (static, walking, indoor, etc) belonging to a specific context type (e.g., mode of movement, indoor/outdoor, or position coordinates). We process the collected traces from each user to produce these snippets. This process must overcome two practical challenges. First, a user may not label a change in context at the precise moment when the change occurs. We drop the first three minutes of a labeled context after a change to increase the likelihood that the context has truly changed. Second, we find that timestamps do not increase monotonically in a trace. We address this issue by making each snippet equal to a maximal sequence of sensor data with increasing timestamps. These snippets are then stitched together. Finally, the stitched trace is used by the simulator which outputs two metrics on completion. We describe the process of stitching next, following by the metrics that we output.

To simulate a more realistic trace, we take snippets, and stitch them together to form

a larger trace. To stitch snippets together, we use a Continuous Time Markov Chain over the state space of all activities. We set up this chain so that the average rate of context changes is 1 every 10 minutes. In reality, transition rates are likely to vary across different activities (for instance, it is more common to transition from static to walking than driving to biking). However, in the absence of detailed data, it is impossible to estimate these transition rates with certainty. Instead, we pick the same transition rate of 1 every 10 minutes for all activities. Next, we simulate this Markov Chain using a specified seed value for a specified duration and look at the sequence of activities and their durations. We use the sequence to pick from our snippets and stitch them together appropriately. We use the stitched trace to simulate the actions of a detector on sensor data. We finally outputs two metrics :

1. **Accuracy:** We measure accuracy as the fraction of time during which the ground truth and the detector agree on an activity. We discretize the ground truth into bins of size L seconds. L is the same as the client specified latency for that detector. For each L second bin, we compute the set of all activities the ground truth saw over that bin, replacing any activity that the client hasn't subscribed to with an X. We repeat a similar discretization procedure for the detector's output. Then for each L second bin in the ground truth, we look at the corresponding bin in the detector's output. If the detector's output is a subset of the ground truth for its corresponding bin, we mark it correct, else we mark it wrong. The accuracy is measured as the fraction of bins where we predict correctly. Unlike prior work, this metric captures what the user really sees when the detector is running. We make the assumption that if the developer specified latency is L , the user does not care when the activity is detected within that time bin of L seconds, so long as it is detected correctly.
2. **Energy Consumption:** We measure energy consumed in Joules over the entire duration of the stitched trace. For each sensor, we compute its energy consumption as $\sum_{i=1}^{i=K} (t_{i+1} - t_i) * P_i$ where P_i is the power consumption of the sensor over the time interval $[t_i, t_{i+1}]$ and K is the number of power consumption switches for sensor j over the duration of the trace. We then add up the energy consumption of all sensors. The power values are derived from hardware models of each phone as described earlier in §3.2.

5.4 Implementation of Related Work

To evaluate the performance of Ada’s mode-of-movement detector, we re-implemented the methods described in UCLA* [48], EEMSS* [57], and SociableSense* [42], and compared their accuracy and energy consumption with Ada’s. This section provides details on our implementations of the three competing methods.

5.4.1 UCLA*

UCLA* uses accelerometer and GPS readings to classify the user’s activity level as stationary, walking, running, biking, or driving. It samples the accelerometer at 32 Hz and the GPS at 1 Hz. Every 1 second, UCLA* computes the mean and the DFT energy coefficient between 1-3 Hz over the time-windowed magnitudes of the three-axis acceleration sensor data. It then combines the accelerometer data with the speed reported from the GPS sensor to construct a feature vector with 5 components.

We used the decision tree classification method, because [48] reports that it achieves the highest accuracy. The classifier was generated using the Weka Machine Learning Toolkit [18], following the methodology described in the UCLA* paper.

We implemented 3 different versions of UCLA*. The first, UCLA*-HMM, implements the algorithm as described in [48]. However, because the original detector was tuned for a fixed detection latency of 1 second, we implemented a second version, UCLA*-MAJ, which takes a majority vote of UCLA*-HMM’s predictions over L seconds. We also implement UCLA*-AVG, which averages the raw feature vectors over a L -second window. However, to make the experiment environment more realistic, we deviated from [48] by skipping the noise filtering that discards GPS readings with low accuracy, and we did not remove slow driving or biking traces.

5.4.2 EEMSS*

EEMSS adopts the strategy of hierarchical sensing to detect if the user is static, walking running, and driving. As shown in Table 5.1, the system maintains a state machine with three states to determine the set of sensors used for classification. In state 1, only accelerometer

State Number	State	Sensors	
1	Static + In a known location	Accelerometer	6 secs sensing + 10 secs sleeping
2	Moving + In a known location	Accelerometer WiFi	6 secs sensing + 10 secs sleeping every 20 secs
3	In a unknown location	GPS	every 20 secs

Table 5.1: The state machine of EEMSS

is powered up. The sensor is used according to a specific duty cycle. EEMSS learns about a user’s movement based on the standard deviation of the three-axis accelerometer magnitude readings. If the accelerometer readings imply that the user is moving, it triggers a state transition from state 1 to state 2.

In state 2, the system uses the WiFi radio to scan for APs, matches the surrounding APs with all the recorded ones, and monitors whether the user has left a known location. The system triggers a state transition from state 2 to state 3 if none of the recorded APs is visible. While in state 2, EEMSS also keeps sampling the accelerometer, to detect if the user has stopped moving. If this happens, the state machine goes back to state 1.

In state 3, EEMSS uses the GPS speed to infer the user’s movement modes. If the GPS reading operation times out (due to satellite signal loss, or because the user has stopped moving for a certain amount of time), the EEMSS state machine goes back to state 2, and resumes scanning for WiFi APs to identify the user’s current location.

In our implementation, we assume that each static trace defines a known location, and maintain the set of APs for each of user’s static trace. To infer user’s movement modes, EEMSS relies on a pre-configured GPS speed range, and on a pre-configured threshold for the standard deviation of the accelerometer’s acceleration vector magnitude. We obtain all the thresholds by finding the values that well separate static, walking, running, and driving in the training set, and used them in the testing phase.

5.4.3 SociableSense*

SociableSense utilizes the accelerometer to distinguish between stationary users and moving users. It proposes an adaptive sampling scheme, which uses a learning technique based on the theory of learning automata to control the accelerometer’s duty cycle. A sensing probability p_i is maintained to determine the duty-cycling interval of the accelerometer. If the sensing

action detects the user is not static, the probability of sensing p_i is increased according to the following formula:

$$p_i = p_{i-1} + \alpha * (1 - p_{i-1}), \text{ where } \alpha = 0.5$$

If the sensing action detects the user is static, the probability p_i is decreased according to the following formula:

$$p_i = p_{i-1} - \alpha * p_{i-1}, \text{ where } \alpha = 0.5$$

Based on the paper, we choose α to be 0.5. We clamped the sensing probability between 0.1 and 0.9, to avoid ending up with a very low sampling rate, which would potentially lead to missing the user’s transitions to the moving state.

When mapping the sensing probability to a duty-cycling interval, the work models the sampling process as a continuous sequence of sense and sleep cycles. A cycle is 3 seconds long. Before the start of each cycle, the system decides whether it will use the accelerometer in this cycle or not based on p_i , i.e. a random sample is drawn from a uniform distribution over $[0, 1)$, and if this sample is less than p_i , then the current cycle becomes a sensing cycle.

SociableSense determines if the user is stationary by comparing the variance of the accelerometer magnitude readings against a pre-configured threshold, which is not specified in [42]. In our implementation, we obtain a threshold by finding the variance boundary value which well separates static and moving in the training set, and use the threshold in the testing phase.

Chapter 6

Results

This chapter presents experimental results of each of Ada’s detectors.

6.1 Mode-of-Movement Detector

In this section, we compare the accuracy and energy consumption of Ada’s mode-of-movement detector with other existing schemes. We also explore the tradeoff between responsiveness and energy consumption, and the relative importance of each feature.

Unless otherwise stated, the experimental setup is as follows, we use two folds from two users (Sensation-1 and Galaxy) as the training set. Each fold has approximately 25 minutes of each of the five modes: static, walking, running, biking and driving. Thus, we have four folds of training data in total. To test different schemes, we use a testing set comprising of four folds of data from two other users (Sensation-2 and Vivid). We also test on two different folds from the training users. In all, we have four test users.

In §4.1.2, we explained how we use KDE to model the distribution of each feature and the effect of picking different variances (h^2). To obtain the most suitable variance for each feature, we do a cross-validation on the training set. We randomly pick one of the two training folds from each training user as the training set, and use the rest of the two folds as the testing set. A grid search algorithm is implemented to evaluate the accuracy performance of each combination of parameters. The best variance we used for each feature is shown in Table 6.1.

Sensor	Feature	KDE Variance
Accelerometer	Mean	0.1
Accelerometer	Standard deviation	0.1
Accelerometer	Peak frequency	0.1
WiFi	Fingerprints similarity	0.3
GPS	Speed	0.5

Table 6.1: The features and their corresponding KDE variances (h^2) used in Ada’s mode-of-movement detector. We use cross-validation on the training set to obtain the best set of parameters, and freeze them before we see the testing data

Note that we freeze these parameters before we see the testing set.

6.1.1 Accuracy and Energy as a Function of Callback Subscription

One of Ada’s features is its ability to adapt to the subscribed modes. For each test user, we run Ada with 31 ($2^5 - 1$) different subscriptions. We stitch four traces; the first two traces are stitched from snippets drawn from Sensation-2 and Vivid and are each eight hours long. The next two traces are stitched from snippets drawn from Sensation-1 and Galaxy and are each four hours long. The total duration of unique traces in both cases is longer than the duration of the stitched trace.

For each of the subscriptions, we plot the energy consumption and accuracy as scatter plots in Figures 6-1 and 6-2. We compare against UCLA* [48]. For all subscriptions, we set the response latency to 15 seconds. We implement 3 different versions of UCLA*. The first, UCLA*-HMM, implements the algorithm as described in [48]. However, because it was tuned for a different latency of 1 second, we implement a second version, UCLA*-MAJ, which takes a majority vote of UCLA*-HMM’s predictions over a 15-second window. We also implement UCLA*-AVG, which averages the raw feature vectors over a 15-second window. We also compare against EEMSS, which does not handle biking. In addition, we compare against SociableSense, which detects only whether the user is static or not, so we evaluate it only on one subscription. Because none of the other schemes adapt to the subscriptions, we represent their energy consumption by constant lines.

The results show that Ada’s mode-of-movement detector is significantly more accurate than all other schemes (in some cases by upto $1.5\times$), while consuming about $4\times$ less energy

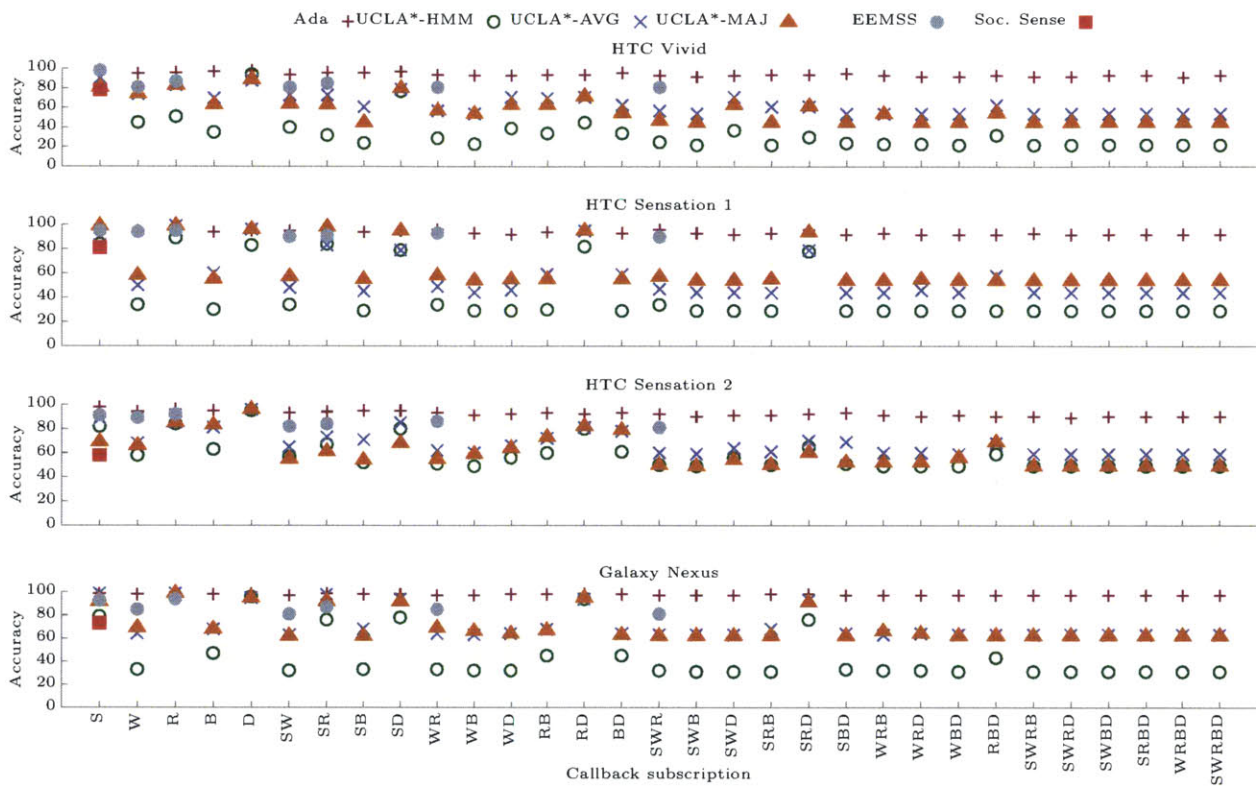


Figure 6-1: Accuracy as a function of subscription. Ada achieves higher accuracy than other schemes always

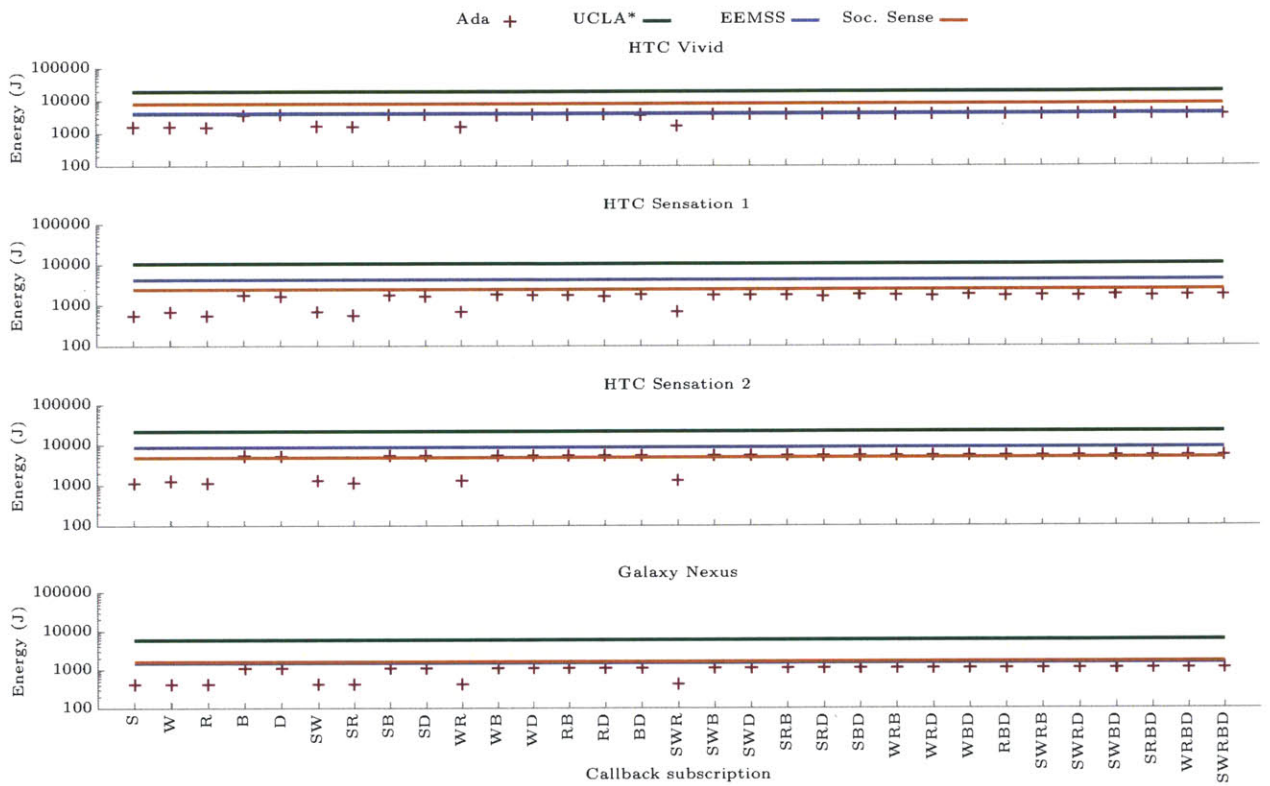


Figure 6-2: Energy consumption as a function of subscription. Ada consumes lower energy than other schemes always

than the UCLA* schemes. The results also show how Ada gains from context-sensitive sensing because it consumes less energy for the static, walking, and running subscriptions compared to the biking and driving modes. Despite detecting many more activities than SociableSense, Ada’s mode-of-movement detector has lower or comparable energy consumption. The main reason why SociableSense performs poorly is because of its long latency of detection. When the sensing probability is small, it requires many cycles to increase the sensing probability. Therefore, it misses the user’s transitions to the moving state.

Ada’s gains in accuracy relative to EEMSS are due to better features leading to better classification; EEMSS’s thresholding approach does not work well. This is best illustrated by looking at specific subscriptions in the graph. On easy-to-distinguish activities such as running and static (R,S,RD,SD), the UCLA* variants and EEMSS achieve accuracy similar to Ada. When subscriptions are harder, such as biking and driving (BD, SBD, SB), Ada’s gains are much more significant.

Ada’s gains in energy consumption relative to the UCLA* variants are due to the sparing use of the GPS. The UCLA* variants all sample GPS frequently. In contrast, Ada’s mode-of-movement detector samples GPS only when the WiFi density is low and completely turns it off while detecting any subset of running, walking or static.

6.1.2 Accuracy and Energy as a Function of Latency

Another feature of Ada is its ability to adapt its sensing strategy to the response latency. We test this by setting the latency to 5, 10, 15, 30 and 60 seconds and plotting the resulting energy and accuracy in Figure 6-3. Ada allows the developer to trade-off promptness of detection for lower energy consumption while maintaining the same accuracy. These energy savings are the result of adapting the WiFi and GPS sampling intervals.

6.1.3 Relative Importance of Different Features

We first evaluate the effectiveness of specific subsets of our features in the mode-of-movement classification problem. For brevity, we consider two extreme cases. The “easy” mode-of-movement classification problem, where the client is only interested in the static callback

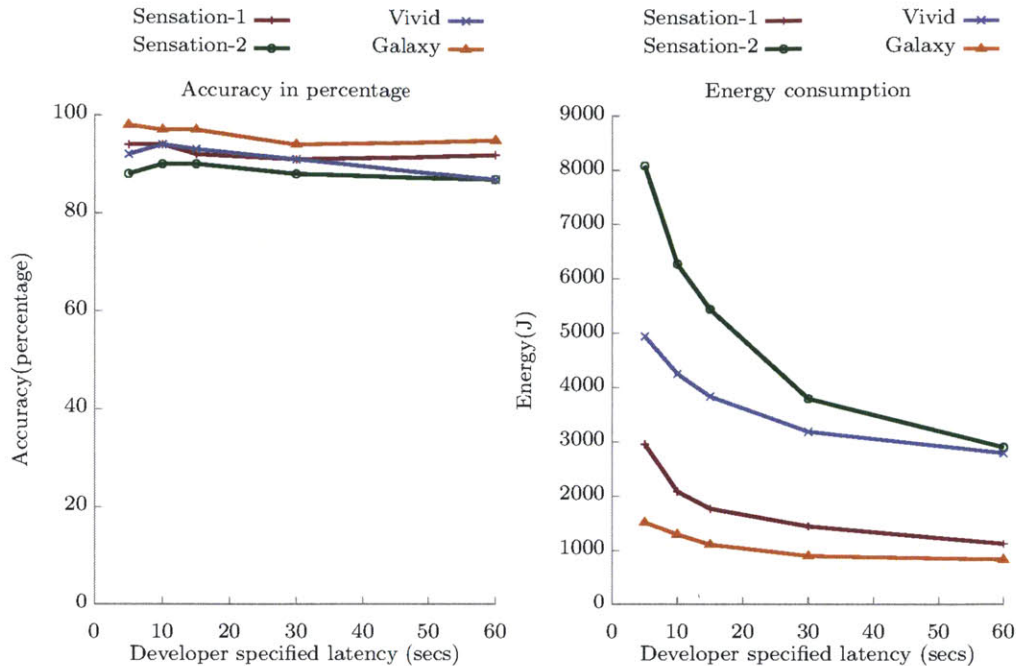


Figure 6-3: Energy and accuracy versus latency. Ada’s mode-of-movement detector trades off latency of detection for decreased energy consumption while maintaining the same accuracy

(Table 6.2), and the “hardest” mode-of-movement classification problem (Table 6.3), where the client is interested in all five modes. We evaluate all seven non-empty subsets of our three accelerometer features: mean, standard deviation, and peak frequency. For each of these seven subsets, we also evaluate the effect of adding WiFi and GPS information to the classification problem. Finally, we evaluate the effect of adding both WiFi/GPS and prediction smoothing on the classification accuracy.

Our results show that for the easy case, it is almost irrelevant which features are used. The difference between the best single feature and combination of features is modest. WiFi/GPS and smoothing do not have any perceivable effect on the classification accuracy. However, for the hardest case, the situation is very different. A combination of two accelerometer features performs better than any single feature, and the combination of all three features performs better than a combination of only two out of three. Furthermore, adding WiFi/GPS improves accuracy from 80% to 85% and smoothing the output improves the accuracy further to 90%. We see similar results for other “hard” callback subscriptions as well (particularly those involving biking).

	M	V	P	MV	VP	MP	MVP
accel only	91	97	91	98	92	97	99
accel, speed	91	97	91	98	92	97	99
accel, speed, ewma	93	97	92	98	97	96	98

Table 6.2: % accuracy with feature subsets, static callback

	M	V	P	MV	VP	MP	MVP
accel only	54	62	57	71	70	72	80
accel, speed	58	61	62	73	74	77	85
accel, speed, ewma	69	69	69	78	86	81	90

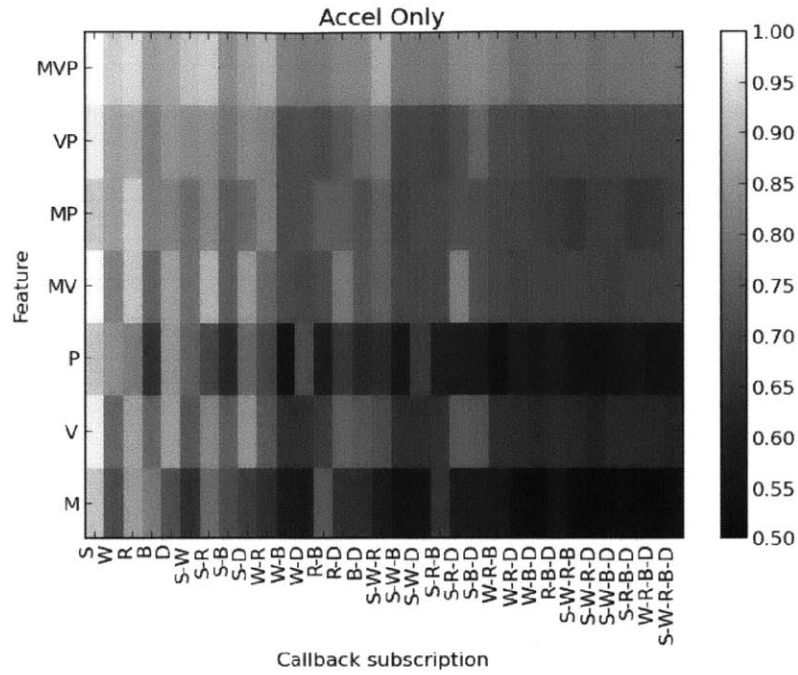
Table 6.3: % accuracy with feature subsets, all 5 callbacks

Another natural question in feature selection is whether any other set of features would work equally well. We answer this question by presenting results (Table 6.4) from five other features from prior work: spectral coefficient at 1 Hz, 2 Hz, and 3 Hz, spectral entropy (SE), and the peak power ratio (PPR). We conduct two sets of experiments with these five new features. In the first set of experiments, we add exactly one of these five features to the already existing mode-of-movement detector in each experiment. We find that for all five experiments, regardless of whether WiFi and GPS are used, the accuracy *reduces* compared to our mode-of-movement detector (MVP). In the second set of experiments, we replace the three acceleration features in our detector with all possible non-empty subsets of these 5 new features. We find that the best detector among the 31 possible subsets (labelled Best of the Rest) still performs poorly in comparison with ours.

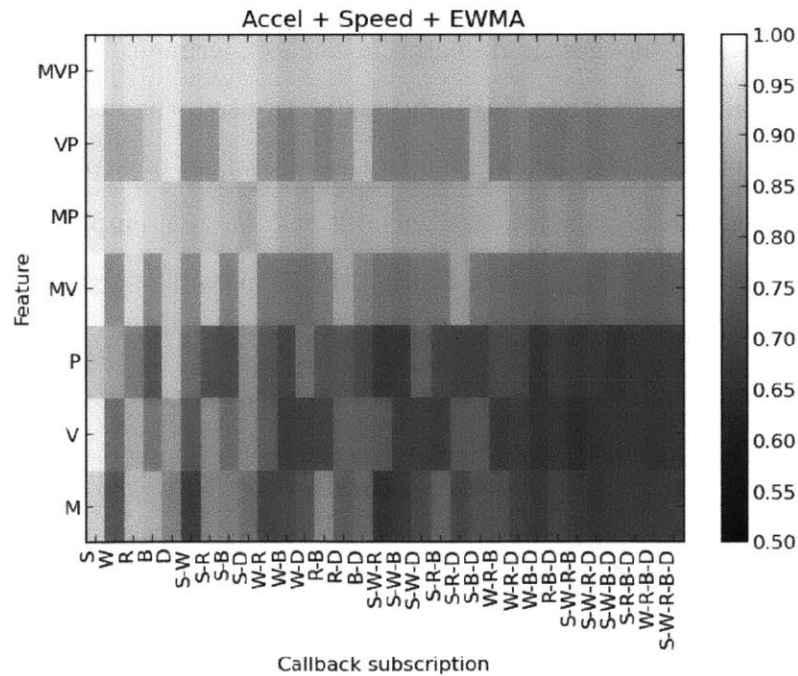
Based on the above results, we might conjecture that our feature set is both minimal and good; adding more features will likely cause overfitting problems.

6.1.4 The Effect of Combining Accelerometer with WiFi/GPS and Smoothing

In this section, we evaluate the performance gain of combining accelerometer with WiFi/GPS and smoothing. Figure 6-4 presents the accuracy for every combination of feature and callback subscription in a heatmap.



(a) Accelerometer only



(b) Accelerometer + WiFi/GPS + EWMA

Figure 6-4: The effect of combining accelerometer with WiFi/GPS and smoothing

	MVP	MVP + 1Hz	MVP + 2Hz	MVP + 3Hz	MVP + PPR	MVP + SE	Best of the rest
accel only	80	78	72	64	63	77	53
accel + speed	85	79	74	69	65	79	55
accel + speed + ewma	90	80	75	70	65	80	56

Table 6.4: % accuracy while adding other features

Latency (seconds)	Mean accuracy & Confidence Interval (percent)	Mean energy & Confidence Interval (Joules)
10	$83.3 \pm 1.0\%$	$4090 \pm 0J$
20	$86.9 \pm 1.1\%$	$2645 \pm 0J$
30	$88.9 \pm 1.0\%$	$2186 \pm 0J$
40	$90.4 \pm 1.3\%$	$1726 \pm 0J$
50	$90.3 \pm 1.0\%$	$1610 \pm 0J$
60	$91.9 \pm 1.3\%$	$1494 \pm 0J$

Table 6.5: Energy and accuracy for indoor-outdoor detector

Figure 6-4a shows that using only the accelerometer, the system is able to detect static, walking, and running with a high accuracy. It also shows that a combination of two accelerometer features outperforms any single feature, and the combination of all three features performs better than a combination of only two out of three; therefore, it is necessary to use all the three features. We also discover that the peak frequency detects walking accurately comparing to other two features. This result is consistent with our observation described in section 4.1.1 that step-based movement modes spike around a certain frequency and the peak frequency captures this feature effectively.

Figure 6-4b shows the effect of incorporating accelerometer data with WiFi/GPS and smoothing. Results shown that the overall accuracy is improved by combining the body movement with speed (Figure 6-4b is brighter than Figure 6-4a).

6.2 Indoor-Outdoor Detector

To collect training and testing data, we alternate walking indoors and outdoors for roughly one minute while tagging the ground truth at all times. We train the detector on about 30 minutes of data covering both indoor and outdoor states collected using a Galaxy Nexus. To test, we collect another 90-minute trace with indoor and outdoor transitions using the same phone. We split this trace into several smaller indoor and outdoor snippets to ensure monotonicity of timestamps. These snippets are then fed into the Stitcher to generate a trace that is three hours long with ten different random seeds. The transition time between

activities in the Stitcher is set to one transition every 10 minutes like before.

We vary the response latency between 10 seconds and 60 seconds. For latencies where the sampling rate $\frac{L}{4}$ isn't one of the WiFi sampling intervals used in our power measurements, we interpolate between the nearest two neighbours. We present results in Table 6.5. The columns are the latency, the average accuracy in percentage and 95% confidence interval across 10 seeds, and the average energy consumption in Joules over the 3-hour stitched trace and 95% confidence interval across 10 seeds.

The table shows how the indoor-outdoor detector effectively allows us to trade off latency of detection for reduced energy consumption and increased accuracy.

6.3 Geofence Detector

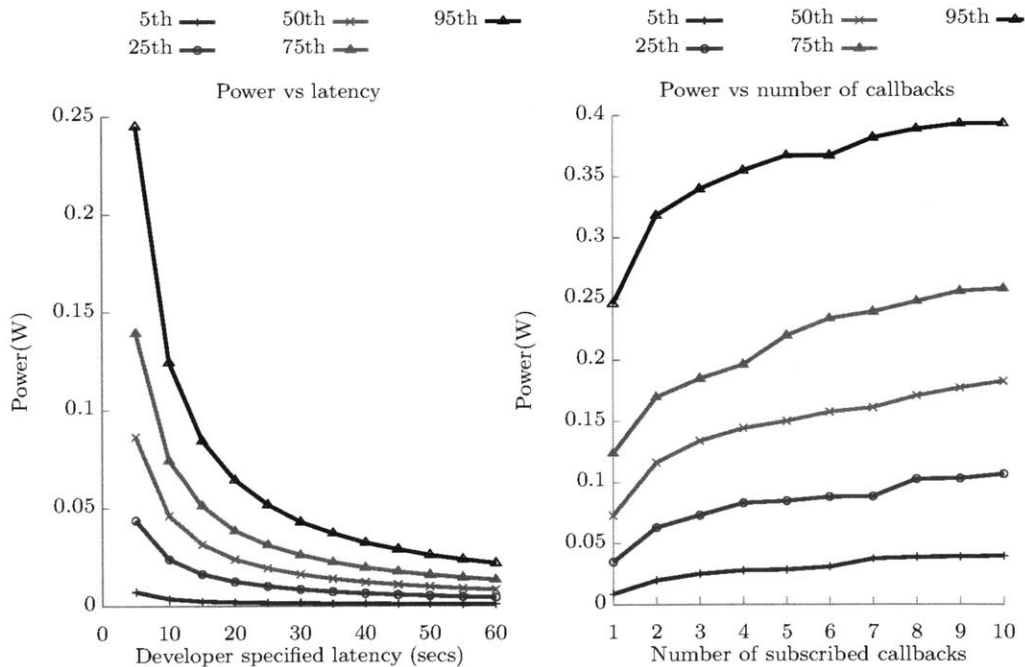


Figure 6-5: Geofence power percentiles. Ada’s geofence detector consumes low energy and its energy consumption increases as the number of geofences increases

We evaluate the geofence detector on 48000 hours of data drawn from 182 users taken from the GeoLife dataset [58, 59]. It consists of 18670 GPS traces (tracks) where a user commutes from a source to a destination. We run a trace driven simulation on this data to

evaluate the accuracy and the energy consumption of our Geofence detector. The location samples on these traces are obtained using GPS. We assume that these samples have high accuracy and hence define the ground truth of the user’s location. To simulate Wifi and GSM sensors at a sampling instant, we add random noise to the GPS location according to the Wifi and GSM accuracy models [54, 55]. We evaluate the accuracy and energy consumption of the Geofence detector in two parts: 1. By varying the latency parameter l and 2. by varying the number of geofences a client subscribes to.

For the first part, we take each trace and assume that a Geofence exists at the destination of the trace. We set the radius of the Geofence to 200 meters, which typically covers a block. As the user commutes from the source to the destination, we simulate our Geofence detector which adapts the sampling time and the sensor used. To estimate the total power consumption, we log the number of times we triggered GPS, Wifi and GSM sensors in our simulation. Using the Power Meter, we estimate the energy for sampling each of these sensors for a specific phone model (Galaxy Nexus) and use these numbers to estimate the energy consumption over the duration of the trace.

Several applications are latency-insensitive and can trade-off promptness of detection for lower energy consumption. Our detector allows this tradeoff by adapting to the latency parameter. As Figure 6-5 (left) shows, the power consumption reduces as the latency parameter increases. Compared to a naive approach that samples GPS at a constant interval, our detector consumes low energy but achieves the same accuracy. For example, using the power consumption values from Figure 3-2, if the GPS on the Galaxy Nexus were to be sampled once every 60 seconds, it consumes around 50 mW, whereas our detector consumes only 8 mW in the median.

Next, we evaluate the energy consumption of our detector by varying the number of geofences the detector looks for. To do this, we pick the top k popular locations for a given user and run the detector on all traces of that user. Figure 6-5 (right) shows that, as the number of geofences increases, the amount of energy consumed increases. It demonstrates how a client can tradeoff the number of geofences it wishes to detect for a decrease in energy consumption.

Chapter 7

Summary and Future Work

In this chapter, we summarize the thesis and suggest possible future research directions.

7.1 Summary and Contributions

In this thesis, we have presented Ada, a context-sensing service for mobile devices. Ada implements three context detectors: (1) mode-of-movement detector, (2) indoor-outdoor detector, and (3) geofence detector. Ada exports a simple API to mobile developers that allows them to control the latency of detection and in turn the accuracy and the energy consumption of the detectors. Within each detector, we propose ideas and observations to tackle the challenges of achieving high accuracy and low energy consumption. In the following, we summarize our main contributions.

In Chapter 3, we present Ada’s system architecture. Ada’s client subscribes to one of several available callbacks from any of Ada’s context detectors with a latency of detection. Ada maintains a list of clients that have subscribed to each of Ada’s context detector, and runs one instance of a specific detector for each phone and sends callbacks from the context detector to their respective clients. Furthermore, the energy consumption of GPS, WiFi, and the accelerometer on three phone models is also presented. We make three key observations that guide the design of Ada’s context detectors. First, there is a clear difference in the power consumption of different sensors. Second, depending on the sampling interval, WiFi scans have a wide range of energy consumption. Third, the accelerometer power consumption

increases with sampling rate despite the fact that accelerometer is itself a low-power sensor.

In Chapter 4, we describe the algorithm of each of Ada’s context detectors. To achieve high accuracy and low energy consumption, the detectors take both the existing context and the desired subscriptions into account, adjusting both the type of sensors and the sampling rates. The main contributions of our mode-of-movement detector over prior schemes are: (1) the use of a new *peak frequency* feature from the acceleration data, used in conjunction with two other known features, (2) a processing algorithm that combines the different features of a feature vector from one sensor using the Naive Bayes classifier, and fuses the outputs of different sensors using *soft voting*, and (3) an adaptive sensor sampling method that takes both the subscribed callbacks and current mode into consideration to reduce energy consumption. Experimental results have demonstrated that the feature set used in the mode-of-movement detector is both minimal and effective; adding more features will cause overfitting problem.

In Chapter 5, we illustrate the rigorous experimental method used in evaluating the performance of Ada. A trace-driven simulator is presented. The simulator takes a sanitized stitched trace as input, and outputs the accuracy and energy consumption of the algorithm. In contrast to previous work, the simulator does not depend on the specific algorithms used in the context detectors.

In Chapter 6, we present the experimental results of each of Ada’s detectors. The main results are as follows:

1. The mode-of-movement detector outperforms previous proposals while consuming between 2 and 3x less energy.
2. The mode-of-movement detector allows developers to trade off latency of detection for lower energy consumption while maintaining the same accuracy.
3. The indoor-outdoor detector allows developers to trade off latency of detection for lower energy consumption and increased accuracy.
4. The geofence detector consumes low energy and its energy consumption increases as the number of geofences increases.

7.2 Future Work

There are several potential directions for future work. First of all, it is of interest to consider the correlation between detectors as opposed to treating each context detector independently. Exploiting the relationships between detectors may provide means to improve detection accuracy and reduce energy consumption. For example, some movement modes can be eliminated if the indoor-outdoor detector deems the user is indoors; the system may save more energy by detecting low energy activities that correlate with the activity that needs to be detected. In addition, implementing the system on mobile devices and evaluating its real-time performance. In contrast to simulating the system offline, implementing a real-time mobile service entails different design considerations, such as the usage of CPU, storage, and memory. Lastly, Ada unleashes the possibility of a wide variety of mobile services and applications. It would be very interesting to extend this work and explore other projects that will enhance the performance of the mobile system.

Bibliography

- [1] Google Activity Recognition API. <http://developer.android.com/reference/com/google/android/gms/location/ActivityRecognitionClient.html/>.
- [2] iOS 6: About Reminders. <http://support.apple.com/kb/HT4970>.
- [3] Locale. <http://www.twofortyfouram.com/>.
- [4] OnX. <https://www.onx.ms/>.
- [5] Tasker. <http://tasker.dinglich.net/>.
- [6] Lauri Aalto, Nicklas Göthlin, Jani Korhonen, and Timo Ojala. Bluetooth and wap push based location-aware mobile advertising system. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, MobiSys '04, pages 49–58, New York, NY, USA, 2004. ACM.
- [7] Ian Anderson and Henk Muller. Practical activity recognition using gsm data.
- [8] Ling Bao and Stephen S. Intille. Activity recognition from user-annotated acceleration data. pages 1–17. Springer, 2004.
- [9] Ling Bao and Stephen S. Intille. Activity Recognition from User-Annotated Acceleration Data. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing*, volume 3001 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2004.
- [10] Yi Chen and Gareth J. F. Jones. Augmenting human memory using personal lifelogs. In *Proceedings of the 1st Augmented Human International Conference*, AH '10, pages 24:1–24:9, New York, NY, USA, 2010. ACM.
- [11] David Chu, Nicholas D. Lane, Ted Tsung-Te Lai, Cong Pang, Xiangying Meng, Qing Guo, Fan Li, and Feng Zhao. Balancing energy, latency and accuracy for mobile sensor data classification. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 54–67, New York, NY, USA, 2011. ACM.
- [12] Ionut Constandache, Shravan Gaonkar, Matt Saylor, Romit Roy Choudhury, and Landon P. Cox. EnLoc: Energy-Efficient Localization for Mobile Phones. INFOCOM '09, pages 2716–2720, 2009.

- [13] James Cussens. Bayes and pseudo-bayes estimates of conditional probabilities and their reliability. In *Proceedings of the European Conference on Machine Learning*, pages 136–152. Springer Verlag, 1993.
- [14] R. Esteller, J. Echauz, T. Tchong, B. Litt, and B. Pless. Line length: an efficient feature for seizure onset detection. In *Engineering in Medicine and Biology Society, 2001. Proceedings of the 23rd Annual International Conference of the IEEE*, volume 2, pages 1707–1710 vol.2, 2001.
- [15] Jonny Farrington, Andrewj. Moore, Nancy Tilbury, James Church, and Pieter D. Biemond. Wearable sensor badge and sensor jacket for context awareness, 1999.
- [16] Raghu K. Ganti, Praveen Jayachandran, Tarek F. Abdelzaher, and John A. Stankovic. Satire: a software architecture for smart attire. In *Proceedings of the 4th international conference on Mobile systems, applications and services*, MobiSys '06, pages 110–123, New York, NY, USA, 2006. ACM.
- [17] Norbert Györbíró, Ákos Fábíán, and Gergely Hományi. An activity recognition system for mobile phones. *Mob. Netw. Appl.*, 14(1):82–91, February 2009.
- [18] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.
- [19] Eija Kaasinen. User needs for location-aware mobile services. *Personal Ubiquitous Comput.*, 7(1):70–79, May 2003.
- [20] J.F. Kaiser. On a simple algorithm to calculate the ‘energy’ of a signal. In *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on*, ICASSP '90, pages 381–384 vol.1, Apr.
- [21] Vaiva Kalnikaite, Abigail Sellen, Steve Whittaker, and David Kirk. Now let me see where i was: understanding how lifelogs mediate memory. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 2045–2054, New York, NY, USA, 2010. ACM.
- [22] Seungwoo Kang, Jinwon Lee, Hyukjae Jang, Hyonik Lee, Youngki Lee, Souneil Park, Taiwoo Park, and Junehwa Song. SeeMon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, MobiSys '08, pages 267–280, New York, NY, USA, 2008. ACM.
- [23] Nicky Kern, Bernt Schiele, and Albrecht Schmidt. Multi-sensor activity context detection for wearable computing. In *In Proc. EUSAI, LNCS*, 2003.
- [24] Donnie H. Kim, Younghun Kim, Deborah Estrin, and Mani B. Srivastava. SensLoc: sensing everyday places and paths using less energy. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 43–56, New York, NY, USA, 2010. ACM.

- [25] John Krumm and Eric Horvitz. Locadio: Inferring motion and location from wi-fi signal strengths. IEEE Computer Society, 2004.
- [26] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. Activity recognition using cell phone accelerometers. *SIGKDD Explor. Newsl.*, 12(2):74–82, March 2011.
- [27] Matthew L. Lee and Anind K. Dey. Lifelogging memory appliance for people with episodic memory impairment. In *Proceedings of the 10th international conference on Ubiquitous computing*, UbiComp '08, pages 44–53, New York, NY, USA, 2008. ACM.
- [28] Lin Liao, Dieter Fox, and Henry Kautz. Learning and inferring transportation routines. *Artificial Intelligence*, page 2007.
- [29] Kaisen Lin, Aman Kansal, Dimitrios Lymberopoulos, and Feng Zhao. Energy-accuracy trade-off for continuous mobile device location. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 285–298, New York, NY, USA, 2010. ACM.
- [30] Jie Liu, Bodhi Priyantha, Ted Hart, Heitor S. Ramos, Antonio A. F. Loureiro, and Qiang Wang. Energy efficient GPS sensing with cloud offloading. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, pages 85–98, New York, NY, USA, 2012. ACM.
- [31] Beth Logan, Jennifer Healey, Matthai Philipose, Emmanuel Tapia, and Stephen Intille. A long-term evaluation of sensing modalities for activity recognition. UbiComp '07, pages 483–500. Springer, 2007.
- [32] Hong Lu, Jun Yang, Zhigang Liu, Nicholas D. Lane, Tanzeem Choudhury, and Andrew 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.
- [33] U. Maurer, A. Smailagic, D.P. Siewiorek, and M. Deisher. Activity recognition and monitoring using multiple sensors on different body positions. In *Wearable and Implantable Body Sensor Networks, 2006. BSN 2006. International Workshop on*, BSN '06, pages 4 pp.–116, April.
- [34] Miluzzo, E. and Lane, N. and Fodor, K. and Peterson, R. and Lu, H. and Musolesi, M. and Eisenman, S. and Zheng, X. and Campbell, A. Sensing meets mobile social networks: The design, implementation and evaluation of the CenceMe application. In *SenSys*, 2008.
- [35] Prashanth Mohan, Venkata N Padmanabhan, and Ramachandran Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. SenSys '08.
- [36] Min Y. Mun, Deborah Estrin, Jeff Burke, and Mark Hansen. Parsimonious mobility classification using gsm and wifi traces. hot-emnets, 2008.

- [37] Suman Nath. ACE: exploiting correlation for energy-efficient and continuous context sensing. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys '12, pages 29–42, New York, NY, USA, 2012. ACM.
- [38] Jeongyeup Paek, Joongheon Kim, and Ramesh Govindan. Energy-efficient rate-adaptive GPS-based positioning for smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 299–314, New York, NY, USA, 2010. ACM.
- [39] Jeongyeup Paek, Kyu-Han Kim, Jatinder P. Singh, and Ramesh Govindan. Energy-efficient positioning for smartphones using Cell-ID sequence matching. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, MobiSys '11, pages 293–306, New York, NY, USA, 2011. ACM.
- [40] Emanuel Parzen. On Estimation of a Probability Density Function and Mode. *Ann. Math. Statist.*, 33(3):pp. 1065–1076, 1962.
- [41] D.J. Patterson, D. Fox, H. Kautz, and M. Philipose. Fine-grained activity recognition by aggregating abstract object usage. In *Wearable Computers, 2005. Proceedings. Ninth IEEE International Symposium on*, ISWC '05, pages 44–51, Oct.
- [42] Kiran K. Rachuri, Cecilia Mascolo, Mirco Musolesi, and Peter J. Rentfrow. Sociable-Sense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing. In *Proceedings of the 17th annual international conference on Mobile computing and networking*, MobiCom '11, pages 73–84, New York, NY, USA, 2011. ACM.
- [43] Cliff Randell and Henk Muller. Context awareness by analysing accelerometer data. Technical report, 2000.
- [44] Matti Rantanen, Antti Oulasvirta, Jan Blom, Sauli Tiitta, and Martti Mäntylä. Inforadar: group and public messaging in the mobile context. In *Proceedings of the third Nordic conference on Human-computer interaction*, NordiCHI '04, pages 131–140, New York, NY, USA, 2004. ACM.
- [45] Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, and Michael L Littman. Activity recognition from accelerometer data. In *Proceedings of the national conference on artificial intelligence*, volume 20 of *IAAI '05*, page 1541. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.
- [46] Lenin Ravindranath, Calvin Newport, Hari Balakrishnan, and Samuel Madden. Improving wireless network performance using sensor hints. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI '11, pages 21–21, Berkeley, CA, USA, 2011. USENIX Association.
- [47] Lenin S. Ravindranath, Arvind Thiagarajan, Hari Balakrishnan, and Samuel Madden. Code In The Air: Simplifying Sensing and Coordination Tasks on Smartphones. Hot-Mobile '12, La Jolla, CA, February 2012.

- [48] Sasank Reddy, Min Mun, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. Using mobile phones to determine transportation modes. *ACM Trans. Sen. Netw.*, 6(2):13:1–13:27, March 2010.
- [49] Murray Rosenblatt. Remarks on Some Nonparametric Estimates of a Density Function. *Ann. Math. Statist.*, 27(3):832–837, September 1956.
- [50] Timothy Sohn, Alex Varshavsky, Anthony LaMarca, Mike Y. Chen, Tanzeem Choudhury, Ian Smith, Sunny Consolvo, Jeffrey Hightower, William G. Griswold, and Eyal de Lara. Mobility detection using everyday GSM traces. In *Proceedings of the 8th international conference on Ubiquitous Computing, UbiComp '06*, pages 212–224, Berlin, Heidelberg, 2006. Springer-Verlag.
- [51] Leon Stenneth, Ouri Wolfson, Philip S. Yu, and Bo Xu. Transportation mode detection using mobile phones and gis information. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '11*, pages 54–63, New York, NY, USA, 2011. ACM.
- [52] Emmanuel Munguia Tapia, Stephen S. Intille, and Kent Larson. Activity Recognition in the Home Using Simple and Ubiquitous Sensors. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing*, volume 3001 of *Lecture Notes in Computer Science*, pages 158–175. Springer, 2004.
- [53] Arvind Thiagarajan, James Biagioni, Tomas Gerlich, and Jakob Eriksson. Cooperative transit tracking using smart-phones. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, SenSys '10*, pages 85–98, New York, NY, USA, 2010. ACM.
- [54] Arvind Thiagarajan, Lenin Ravindranath, Katrina LaCurts, Samuel Madden, Hari Balakrishnan, Sivan Toledo, and Jakob Eriksson. VTrack: Accurate, Energy-Aware Road Traffic Delay Estimation Using Mobile Phones. *SenSys '09*.
- [55] Arvind Thiagarajan, Lenin S. Ravindranath, Hari Balakrishnan, Samuel Madden, and Lewis Girod. Accurate, Low-Energy Trajectory Mapping for Mobile Devices. In *8th USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, NSDI '11, Boston, MA, March 2011.
- [56] Tim van Kasteren, Athanasios Noulas, Gwenn Englebienne, and Ben Kröse. Accurate activity recognition in a home setting. In *Proceedings of the 10th international conference on Ubiquitous computing, UbiComp '08*, pages 1–9, New York, NY, USA, 2008. ACM.
- [57] Yi Wang, Jialiu Lin, Murali Annavaram, Quinn A. Jacobson, Jason Hong, Bhaskar Krishnamachari, and Norman 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, MobiSys '09*, pages 179–192, New York, NY, USA, 2009. ACM.

- [58] Yu Zheng, Yukun Chen, Quannan Li, Xing Xie, and Wei-Ying Ma. Understanding transportation modes based on GPS data for web applications. *ACM Trans. Web*, 4(1):1:1–1:36, January 2010.
- [59] Yu Zheng, Like Liu, Longhao Wang, and Xing Xie. Learning transportation mode from raw GPS data for geographic applications on the web. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 247–256, New York, NY, USA, 2008. ACM.
- [60] Zhenyun Zhuang, Kyu-Han Kim, and Jatinder Pal Singh. Improving energy efficiency of location sensing on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 315–330, New York, NY, USA, 2010. ACM.