Software Techniques in the Compromise of Energy and Accuracy

by

Jeffrey Michael Boyd

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved April 2014 by the
Graduate Supervisory Committee:

Hari Sundaram, Co-Chair
Baoxin Li, Co-Chair
Aviral Shrivastava
Pavan Turaga

ARIZONA STATE UNIVERSITY

May 2014

ABSTRACT

Software has a great impact on the energy efficiency of any computing system–it can manage the components of a system efficiently or inefficiently. The impact of software is amplified in the context of a wearable computing system used for activity recognition. The design space this platform opens up is immense and encompasses sensors, feature calculations, activity classification algorithms, sleep schedules, and transmission protocols. Design choices in each of these areas impact energy use, overall accuracy, and usefulness of the system.

This thesis explores methods software can influence the trade-off between energy consumption and system accuracy. In general the more energy a system consumes the more accurate will be. We explore how finding the transitions between human activities is able to reduce the energy consumption of such systems without reducing much accuracy. We introduce the Log-likelihood Ratio Test as a method to detect transitions, and explore how choices of sensor, feature calculations, and parameters concerning time segmentation affect the accuracy of this method. We discovered an approximate 5× increase in energy efficiency could be achieved with only a 5% decrease in accuracy.

We also address how a system's sleep mode, in which the processor enters a low-power state and sensors are turned off, affects a wearable computing platform that does activity recognition. We discuss the energy trade-offs in each stage of the activity recognition process. We find that careful analysis of these parameters can result in great increases in energy efficiency if small compromises in overall accuracy can be tolerated. We call this the "Great Compromise." We found a 6× increase in efficiency with a 7% decrease in accuracy.

We then consider how wireless transmission of data affects the overall energy efficiency of a wearable computing platform. We find that design decisions such as

feature calculations and grouping size have a great impact on the energy consumption of the system because of the amount of data that is stored and transmitted. For example, storing and transmitting vector-based features such as FFT or DCT do not compress the signal and would use more energy than storing and transmitting the raw signal. The effect of grouping size on energy consumption depends on the feature. For scalar features energy consumption is proportional in the inverse of grouping size, so it's reduced as grouping size goes up. For features that depend on the grouping size, such as FFT, energy increases with the logarithm of grouping size, so energy consumption increases slowly as grouping size increases.

We find that compressing data through activity classification and transition detection significantly reduces energy consumption and that the energy consumed for the classification overhead is negligible compared to the energy savings from data compression. We provide mathematical models of energy usage and data generation, and test our ideas using a mobile computing platform, the Texas Instruments Chronos watch.

For Michelle. This would not have been possible without your love and patience.

Thank you.

I love you.

ACKNOWLEDGEMENTS

I'd like to first thank my mentor and advisor, Hari Sundaram, for working with me on these ideas for the past few years. Thank you for opening the wide world of university research to me with its many disciplines and for your support when the going got tough.

This work would not have been possible without the resources and motivation of the Mixed Reality Rehabilitation lab, part of the School of Arts, Media, and Engineering at Arizona State University. This lab is the reason I came to ASU in the first place. I learned early in my life that I loved solving technical problems. As a young adult I realized I love working with people in need. The Mixed Reality Rehabilitation lab was an ideal setting to satisfy both those desires. I am grateful for the people I worked closely with in the lab: Hari Sundaram, Thanassis Rikakis, Loren Olsen, Yinpeng Chen, Margaret Duff, Michael Baran, Nicole Lehrer, Diana Siwiak, and Assegid Kidane.

I gratefully acknowledge the financial support from Science Foundation Arizona, and National Science Foundation Integrative Graduate Education and Research Traineeship (NSF IGERT), under NSF grant DGE-05-04647. This support allowed me to support my growing family with zero student loan debt.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

I want to begin with a quote from Professor Steve Furber, co-designer of the ARM microprocessor:

> If you want an ultimate low-power system, then you have to worry about energy usage at every level in the system design, and you have to get it right from top to bottom, because any level at which you get it wrong is going to lose you perhaps an order of magnitude in terms of power efficiency. The hardware technology has a first-order impact on the power efficiency of the system, but you've also got to have software at the top that avoids waste wherever it can. You need to avoid, for instance, anything that resembles a polling loop because that's just burning power to do nothing. [Furber and Brown, 2010]

Steve Furber is one that is thoroughly familiar with the electrical properties of computing systems and has been an active contributer to the field since the early 1980s. Therefore, his exclamation that top-level software should avoid waste wherever it can should be taken seriously. Software can have an order of magnitude effect on the energy usage of a system; Dr. Furber alludes to this in the above quote and this thesis will demonstrate evidence of this. Software affects many levels of the system design, so efficiencies and inefficiencies have system-wide ripple effects.

The effect of software on energy efficiency is especially interesting in the context of mobile computing platforms. These devices are a hotbed of research and development. In general they contain a low-power processor, a small amount of non-volatile memory,

one or more sensors, a radio transceiver, and, perhaps most importantly, get their energy from a battery. This combination of features on a single platform provide many opportunities to make decisions on compromises between energy consumption and accuracy. For example, a Global Positioning System (GPS) sensor is very accurate at determining location, but it consumes more power than using cellular triangulation, which also provides location, but is much less accurate. Consider also the design decision of what size battery to use on a wearable device. A larger battery will mean the system can run longer before recharging, but a larger battery takes up more space and is heavier than a smaller one–possibly adversely affecting the person wearing the device. An energy-constrained device demands efficient use of resources, and software manages those resources.

1.1 Motivation

This thesis is further motivated by the research into rehabilitation systems for stroke survivors conducted in the Mixed-Reality Rehabilitation Lab, part of the School of Arts, Media, and Engineering at Arizona State University. Stroke survivors are interesting subjects as part of a study on how software affects mobile computing platforms. A stroke usually impairs one side of a person's body, so a key indicator of rehabilitation or progress is a person's ability to perform routine activities of daily living (ADL) with their impaired side. ADL are routine functions such as brushing teeth, reaching for objects, using eating utensils, opening and closing doors or drawers, etc. Physical and occupational therapists are keen to know if their patients use their impaired side for ADL, but currently rely on subjective, after-the-fact reports and surveys from patients.

A more objective measurement would be useful in getting a more complete understanding of a stroke survivor's recovery. A wearable computing platform could

provide a more objective measure if it was as small and unobtrusive as a wristwatch, comfortable enough for a person to wear throughout the day, able to identify activities of daily living, and work without frequent recharges to the battery or battery replacements. It was in the context of this situation that we decided to explore how our knowledge of human behavior could help the overall system be more energy efficient.

A key insight was the observation that humans move and work at completely different time scales than what modern microprocessors operate at. Microprocessors perform millions of operations per second, but humans do activities on scales of tens of seconds, minutes, and hours. Software managing a mobile computing platform could exploit this knowledge and remain in a low-power state until a person starts a new activity. The transitions between activities are the interesting points in time to detect and are key in reducing the energy consumption of a wearable device.

1.2 Scope

I think it's important to identify some aspects of energy efficiency that are not covered in this thesis. No discussion on energy efficiency would be complete without at least mentioning Jevons Paradox. William Stanley Jevons was a 19th century English economist who observed that increases in the efficiency of coal burning from James Watt's steam engine resulted in greater consumption of coal, rather than a decrease in consumption. Some of his peers had argued that increases in the efficiency of coal burning would decrease the consumption of this constrained resource, but Jevons argued the contrary was true.

Parallels can be made with efforts to increase the energy efficiency of software. Indeed, advances in the capabilities of low-power processors, thanks to Moore's Law, have kicked off a surge in hype surrounding the "Internet of Things," which are networked, low-power systems embedded in common physical object that have not

commonly been augmented with electronics. Gartner, Inc. projects the Internet of Things will grow to 26 billion devices by the year 2020 [Rivera and van der Meulen, 2013]. This thesis will not address the business and societal consequences of Jevons Paradox as it applies to software for embedded devices, other than pointing out that any increases in energy efficiency software can provide would be substantial when applied across 26 billion devices.

This thesis will not address active research into activity or pattern recognition, machine learning, or artificial intelligence, though concepts from these disciplines will be mentioned and used. Nor will it address advances in digital logic design or silicon processes that enable greater energy efficiency from a hardware perspective.

Given the idea of using our knowledge of human behavior to make a wearable computing platform for stroke survivors more energy efficient, and the idea of finding transitions between activities, we will explore the design space this problem opens up. We introduce the Log-likelihood Ratio Test as a method to detect transitions, and explore how choices of sensor, feature calculations, and parameters concerning time segmentation affect the accuracy of this method. We discovered an approximate $5\times$ increase in energy efficiency could be achieved with only a 5% decrease in accuracy.

We also address how a system's sleep mode, in which the processor enters a low-power state and sensors are turned off, affects a wearable computing platform that does activity recognition. We discuss the energy trade-offs in each stage of the activity recognition process. We find that careful analysis of these parameters can result in great increases in energy efficiency if small compromises in overall accuracy can be tolerated. We call this the "Great Compromise." We found a $6\times$ increase in efficiency with a 7% decrease in accuracy.

We then consider how wireless transmission of data affects the overall energy efficiency of a wearable computing platform. We find that design decisions such as

feature calculations and grouping size have a great impact on the energy consumption of the system because of the amount of data that is stored and transmitted. For example, storing and transmitting vector-based features such as FFT or DCT do not compress the signal and would use more energy than storing and transmitting the raw signal. The effect of grouping size on energy consumption depends on the feature. For scalar features energy consumption is proportional in the inverse of grouping size, so it's reduced as grouping size goes up. For features that depend on the grouping size, such as FFT, energy increases with the logarithm of grouping size, so energy consumption increases slowly as grouping size increases.

We find that compressing data through activity classification and transition detection significantly reduces energy consumption and that the energy consumed for the classification overhead is negligible compared to the energy savings from data compression. We provide mathematical models of energy usage and data generation, and test our ideas using a mobile computing platform, the Texas Instruments Chronos watch.

1.3 Thesis Roadmap

Chapter 2 discusses work related to this thesis. The effects of software on the energy consumption of a system has been known for quite some time; early work concentrated on general software optimizations that reduced power. It wasn't until recent years that researchers looked specifically at mobile computing platforms and looked into ideas to reduce their energy consumption based on their purpose. Chapter 3 introduces and explores the design space surrounding our first transition detection method, the Log-likelihood Ratio Test. Chapter 4 expands on the work in Chapter 3 by introducing two other transition detection methods and explores the effects of different sleep patterns. Chapter 5 expands the design space to include the energy

consumed by wireless transmission of data and highlights the importance of data compression. It also introduces mathematical models of energy consumption and data generation for different system levels.

Chapter 2

RELATED WORK

2.1 Early Work

Early work on how software affects energy consumption identified three hardware subsystems that software can significantly influence, namely the memory system, system buses, and data paths to arithmetic logic units and floating-point units [Roy and Johnson, 1997; Tiwari *et al.*, 1996]. They discuss how to estimate the energy consumption of a particular program and describe several techniques to optimize software to use less power, mostly through code optimizations, taking advantage of hardware low-power options and caches, and exploiting parallelism when available. One important insight is that the shortest code sequences resulted in the lowest program energy consumption. This important work led to an awareness of the impact software has on the energy consumption of systems, opening the door for hardware-software co-design and helped compiler designers write compilers that generate more energy efficient code.

Expanding on the previous work, which was done mainly for desktop computers, several papers addressed low power principles for entire mobile, battery-operated systems such as laptop computers and cellular phones [Smit and Havinga, 1997; Lorch and Smith, 1998; Havinga and Smit, 2000]. They address the device as an ecosystem of processing units, input-output devices, networking devices and device operator. Important insights for hardware include reducing the operating voltage and switching frequency, designing components that have low-power modes with reduced functionality, and having dedicated subunits that are more power-efficient than a general

purpose component. Also, adding some complexity, such as adding cache, does reduce power consumption because it reduces the number of (relatively power-hungry) memory accesses. Trade-offs have to be made as well, since the most power-efficient systems are often the least flexible, and the user should be given some control over the power response of the system. Any evaluation of an energy saving strategy should include whether the trade-offs are desirable for the users.

Zotos et al. proposed using a new measurement, energy complexity, in addition to time and space complexity, as a performance measurement of algorithms [Zotos *et al.*, 2005]. They model the energy consumed by the system to CPU instructions, memory accesses to instructions and memory accesses to data. They demonstrated the usefulness of their model with three different implementations of matrix multiplication. Though their model overestimated energy consumption, it was consistent in the amount it overestimated. Their model only considers computation and assumes a non-mobile platform. We use a similar model for energy consumption in a mobile computing platform, but include more components usually found in those platforms.

2.2 Activity Recognition with Inertial Sensors

The use of inertial sensors, such as accelerometers and gyroscopes, in monitoring and classifying a wide variety of activities from healthcare to sports is a well-studied subject. The work can roughly be divided into two camps: those that focus on recognition accuracy and those that take energy consumption into account.

2.2.1 Work that focuses on recognition accuracy

Accelerometers have been shown to be a valid and objective measurement of impaired arm usage in stroke survivors [Uswatte *et al.*, 2006]. More recently, accelerometers have been used to accurately estimate the Functional Ability Scale (FAS) in

stroke survivors [Patel *et al.*, 2009]. The FAS is a clinical scale that assesses the function and impairment of stroke survivors. Accelerometers have been used to identify routines throughout the day [Huynh *et al.*, 2008] and been combined with RFID readers to monitor daily activities and interaction with devices tagged with RFID tags [Stikic *et al.*, 2008]. One thing these and many other studies have in common is their lack of concern for the energy consumption of wearable devices. Indeed, activity recognition projects have traditionally focused on obtaining the highest possible accuracy and avoiding overfitting rather than energy efficiency.

For more information on accelerometers for activity recognition and healthcare, see a survey by Gebruers *et al.* [2010]. They focus on clinical trials that used accelerometers to measure the physical activity of stroke survivors. Key findings from their survey are that patients were compliant and motivated to wear motion loggers, and devices worn on the wrist, hip, and ankle were perceived to be non-restricting.

Activity recognition, including gesture spotting, is a popular area of research with many published papers on the subject. Some recent examples include Czabke *et al.* [2011], who developed a highly accurate accelerometer-based activity recognition system that generalizes across users and sensor orientations. They limit the activities to resting, walking, running, and unknown activity. They do not discuss the energy consumed by their implementation or possible ways to conserve energy. Also, Krassnig *et al.* [2010] created a very accurate activity classifier that generalizes across users. They use a single 3-axis accelerometer near the center of mass. They do not consider how much energy their system consumes.

A good survey of current practices and open research problems was written by Avci *et al.* [2010]. Avci et. al. report on the various techniques in the published literature used in the main steps of activity recognition: preprocessing, segmentation, feature extraction, and classification. One conclusion they draw is that there is a

growing problem of minimizing communication and energy consumption, two areas of focus in this thesis.

2.2.2 Work that considers energy consumption

There have been a few studies that recognize that there is a trade-off between activity recognition accuracy and energy efficiency. Stäger, Bharatula and others have presented an empirical design methodology to explore the trade-offs between energy and accuracy in a wrist-worn system consisting of accelerometers, microphone, and light sensors [Stäger *et al.*, 2007; Bharatula *et al.*, 2005]. They took a low-power approach from the beginning, investigating and showing how sampling frequency, feature selection, and choice of classifier change their system's power consumption. An important finding was that a lot of battery life could be gained by sacrificing a little accuracy. Another insight was that the addition of a sensor could improve the energy efficiency [Bharatula *et al.*, 2005]. Both of these projects focus on the sampling frequency of the sensors and feature selection as means of reducing energy while maintaining a regular, periodic sleep schedule. Not only do we use sampling frequency and feature selection to reduce energy, we explore different sleep strategies.

Another study recognized the trade-off between power and accuracy. Krause and others used the accelerometers on the eWatch system to classify five activities: walking, running, standing, sitting and climbing or descending stairs [Krause *et al.*, 2005]. They showed a 4× increase in the lifetime of their wrist-mounted system, without significantly reducing prediction accuracy, by reducing the sampling frequency and exploring different sleep schedules. They refer to these sleep schedules as selective sampling strategies. Their best results came from a sampling strategy that relies on extensive training data and a model of activity sequences with probabilities of transitioning between activities, which they tested against a strategy based on exponential

back-off, a strategy based on sleep times drawn from a uniform distribution, and the baseline uniform sampling with regular periodicity.

French et al. expanded on the work of Krause by collecting more data and focused specifically on evaluating different selective sampling strategies [French *et al.*, 2007]. They tested a baseline uniform sampling strategy, a strategy that samples over the distribution of duration times of activities (assuming a Gaussian distribution), and one that samples based on the probability of a transition occurring. Our work in activity transition detection complements the selective sampling strategies described by Krause et al. and French et al. Also, their methods rely on extensive *a priori* knowledge about the duration of activities and probabilities of transitions. Our methods assume no prior knowledge. Further, we explore a much larger design space beyond just the sampling rate of the sensors.

Another project studied how changing the sleep schedule affects power and accuracy in recognizing activities [Au *et al.*, 2009]. Their algorithm, called Episodic Sampling, uses a method similar to the exponential back-off strategy employed by Krause et al. They report dramatic decreases in energy consumption, data storage needs, and large gains in battery life while only sacrificing approximately 5% in average accuracy compared to a continuous classification strategy. Episodic Sampling will be discussed in more detail in Section 4.2.2.

A different approach to reducing power in an activity recognition system was taken by Wang et al., who focused on using the accelerometers, microphone, wi-fi, and GPS radios in a smartphone [Wang *et al.*, 2009]. They reduce power with a sensor hierarchy and decision tree that ensure only the minimum number of sensors are active at any time and that lower-power sensors activate higher-power sensors only when necessary. They further reduce energy consumption by manually setting sampling strategies for each sensor, based on extensive training data. In this paper,

we attempt to automatically set the sampling strategy, without the need of training data.

There has been recent work that recognizes how software can improve the energy efficiency of wearable systems. Sun *et al.* [2011] recognize that energy optimizations are possible even when continuously monitoring physiological signals. They use accelerometers to measure activity levels and optimize the settings of an electrocardiogram (ECG). They take advantage of using a low-power microcontroller to calculate features on the device itself, thus reducing the overhead of wireless transmission. This extends the battery life 2.5 times.

Raffa *et al.* [2010] present a method of recognizing specific gestures from continuous accelerometer and gyroscope data in an energy-efficient manner. They leverage relatively simples features and algorithms calculated on a wearable device to distill the stream of data to the most essential parts. They do this to limit the times computationally-complex activity classification algorithms are called, which also reduces overall system power without sacrificing much accuracy. They wirelessly transmit the raw data, probably because the features they use actually generate more data than the raw data. They focus on high-accuracy, with very few false positives, because their system is designed to be an input to a computer's user interface.

Patel *et al.* [2009] conserve energy in an electroencephalogram (EEG) based seizure detection system by downsampling the EEG sampling frequency, reducing the number of bits to represent data, and using hardware approximations of features. They also simulate how their system would perform on a low-power digital signal processor and custom, application-specific circuit. They also find that transmitting all EEG data consumes the most energy and that reducing the amount of data that is transmitted significantly extends the lifetime of their device.

Chapter 3

TRANSITION DETECTION

We introduce the idea of transition detection as one method of reducing the energy consumption of wearable computing platform that does activity classification. Activity classification or pattern recognition algorithms are computationally complex and the overall system would be inefficient if it were to continuously run these algorithms. It would be accurate, but inefficient because the time scale at which people change activities is far greater than the time scale it takes for a computer to run a classification algorithm. The ideal would be for the classification algorithms once per activity and not run again until the next activity. The overall system would be more efficient if we had some method of detecting transitions that works well and is less complex than pattern recognition algorithms.

This chapter explores the questions that came out of this idea. We first explore the design space created by the problem of transition detection. Figure 3.1 outlines the general procedure of our transition detection system. Then we explain the log-likelihood ratio test we developed as a way to detect transitions. Next we develop ways to evaluate our transition detection scheme and explain the experimental setup we used to test it. Last we summarize the findings.

3.1 Design Space

In the following sections we outline the parameters of our design space in three broad categories: sensors, features, and temporal resolution. We define temporal resolution as the various time-based controls (sampling frequency, window duration, etc.) we have in the system.

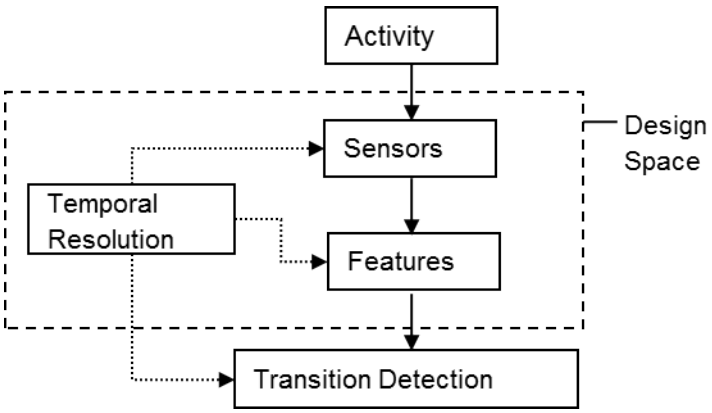

Figure 3.1: The design space in low-power activity transition detections focuses on the sensors that detect activities, the features extracted from these signals, and the temporal resolution or time-dependent properties of the system.

3.1.1 Sensors

To sense the activities we are interested in we chose to use a wrist-mounted triaxial accelerometer. Accelerometers offer several advantages, beginning with the fact that they're small, lightweight, and inexpensive. Accelerometers are also widely used in the literature on wearable computing systems and human activity recognition [Bao and Intille, 2004]. We also investigated using magnetometers and gyroscopes. We found the magnetometers in our test system to be too noisy for any practical use and preliminary tests that included gyroscopes yielded poor results.

We used a triaxial accelerometer and experimented with all seven possible combinations of signals, x-axis; y-axis; z-axis; x and y axes; x and z axes; y and z axes; x, y, and z axes.

3.1.2 Features

Features are some aspect or quantitative measurement of the signal. They can be simple time-domain measurements such as maximum, minimum, variance, and mean, or frequency-domain based such as the Fast Fourier Transform (FFT) and

the Discrete Cosine Transformation (DCT). Other projects have had success using wavelet transformations [Nyan *et al.*, 2006; Sekine *et al.*, 2000]. Each feature has its own computational complexity, summarized in Table 3.1. Computational complexity of feature extraction is important because it is directly related to energy consumption [Zotos *et al.*, 2005].

3.1.3 Temporal Resolution

The third variable is sampling frequency. We chose 100Hz as a baseline. We chose this value because the fastest hand movements are about 5 Hz, and a good rule-of-thumb is to oversample about 20x when using a noisy sensor. Realizing there are low-power advantages to sampling at lower frequencies and encouraged by the good results of Krause *et al.* [2005], who sampled at much lower frequencies, we also experimented sampling at 50, 20, and 10 Hz. Lower sampling frequencies mean fewer samples to process, faster runtimes, and increased energy savings.

The fourth variable is the size of the observation. We call this observation a frame and define it as the number of samples that the features are extracted from. In our study we used frame sizes of 10 and 20 samples.

Last we have the length or duration of the sliding window, measured in seconds. The length of the window affects the number of observations used to calculate the likelihood function, as well as the total number of comparisons. In our study we used window lengths of 6, 8, 10, 12, 14, 16, 18, and 20 seconds.

There are 4480 permutations of the variables in our design space.

3.2 Transition Detection Using a Log-Likelihood Ratio Test

Our transition detection method uses a measurement of how different two sections of the signal are within a bounded window of time. More specifically, given some point

Table 3.1: This table is adapted from Avci et. al's survey on using inertial sensors for activity recognition in the healthcare industry [Avci *et al.*, 2010]. In terms of energy consumption, features can be characterized by their computational and spatial complexity. Those features with complexity $O(N \log N)$ will take longer to compute and consume more energy than those with complexity $O(N)$. Similarly, vector features will consume more energy than scalar ones because of the increased amount of data, especially when algorithms that use the data later in the system are $O(N^3)$ (such as matrix inversions).

Feature	Complexity	Scalar or Vector
Mean	$O(N)$	S
Variance/Std. Dev.	$O(N)$	S
RMS	$O(N)$	S
Cum. Histogram	$O(N)$	S
Zero/Mean Crossing Rate	$O(N)$	S
Derivative	$O(N)$	V
Peak Count	$O(N)$	S
Sign	$O(1)$	S
Spectral Centroid	$O(N)$	S
Spectral Energy	$O(N)$	S
Spectral Entropy	$O(N)$	S
Wavelet Coefficent	$O(N)$	V
Signal Magnitude Area	$O(N)$	S
Signal Magnitude Vector	$O(N)$	V
Inter-axis Correlation	$O(N)$	S
Freq. Range Power	$O(N \log N)$	S
Fast Fourier Transform	$O(N \log N)$	V
Discrete Cosine Transform	$O(N \log N)$	V

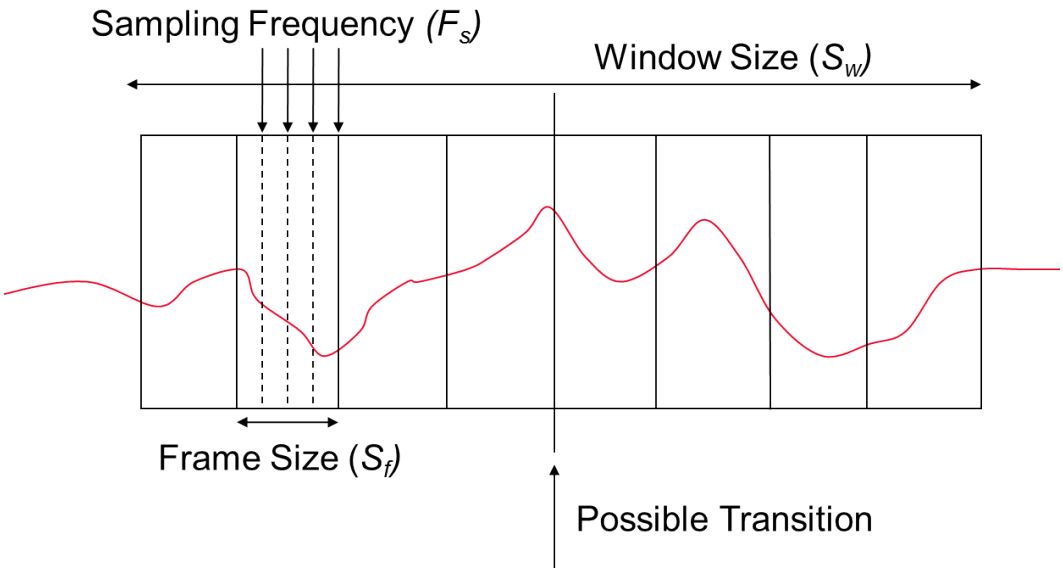

Figure 3.2: The window is divided into left and right panes and compared using a log-likelihood ratio test. The window panes are made of several frames, which are comprised of several samples (shown in dashed lines). Features are calculated per frame and can be a scalar or a vector.

in time, we want to measure how likely it is we have a transition there by looking at the window of time immediately before and after that point in time. We split the window in half, creating left and right window panes, as seen in Figure 3.2. We want to measure how different the panes are from each other and the entire window. First we define the log-likelihood function. For each of these panes and for the entire window itself, we calculate a log-likelihood function for each signal we are analyzing:

$$L\left(x_1, \ldots, x_N\right) = \sum_{i=1}^{N} \ln\left(p\left(x_i|\mu, \Lambda\right)\right) \tag{3.1}$$

In Equation 3.1, $x_1, \ldots, x_N$ denote the N observations or frames from the left pane, right pane or the whole window. The values μ and Λ are the mean feature vector and covariance matrix of all N observations. The probability p is derived from the multivariate Gaussian probability distribution function. Each observation x_i is a vector of features extracted from each signal.

Once these likelihoods have been calculated for each part of the window, we com-

bine them in a log-likelihood ratio test, seen in Equation 3.2.

$$LLRT_i = \frac{L\left(i, \ldots, i + \frac{N_f}{2}, j, \ldots, j + \frac{N_f}{2}\right)}{L\left(i, \ldots, i + \frac{N_f}{2}\right) + L\left(j, \ldots, j + \frac{N_f}{2}\right)} \tag{3.2}$$

In Equation 3.2, the variable i represents the first frame of the left window pane, j is the first frame of the right window pane, and N_f is the total number of frames per window. The intuition behind the LLRT is that it analyzes if the two distributions are identical. The LLRT will be close to one when no transition is present and greater than one when a transition is present. It peaks where the probability of a transition is greatest.

The LLRT can become computationally complex when the dimension of the feature vector is high. This happens when using multiple scalar features or the coefficients of a Fourier, Discrete Cosine, or wavelet transformation. The complexity of the LLRT is proportional to the square and cube of the dimension of the feature vector. This is because of the calculation of the covariance matrix and its inverse, which are $O(N^2)$ and $O(N^3)$ functions, respectively. This is discussed further in Section 3.3.2 and Appendix A.

3.3 Evaluation Metrics

We evaluate how good $LLRT$ is at detecting transitions using precision, recall, F-score and estimated energy consumption based on the runtime of the experiments.

3.3.1 Accuracy

Our accuracy measurements are based on hits, misses and false positives. When there is a true transition, the transition detector can either correctly detect the transition (hits) with probability P_C, or miss the transition with probability P_M. When there is no transition, the transition detector can incorrectly raise a false alarm with

probability P_{FA}. We then combine these into precision, and recall, where

$$Precision = \frac{P_C}{P_C + P_{FA}}$$

and

$$Recall = \frac{P_C}{P_C + P_M}$$

A common measure that combines both is the F-Score:

$$F = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

F is equal to 1 when both $Precision$ and $Recall$ are 1. We also define Reverse F-Score (RF) measure:

$$RF = 1 - F$$

which reverses the F-Score so that 1 is bad and 0 is good. We use RF to more easily visualize accuracy vs. runtime.

3.3.2 Computational Complexity

We estimate energy as a computational complexity metric, since to a first order, energy consumption is very strongly correlated to computational complexity. This is because computational complexity directly affects runtime and runtimes affect energy consumption. We've developed a model for computational complexity based on the variables in our system.

The computational complexity, C, of our system can be broken up into two parts: feature extraction (C_{FE}) and the log-likelihood ratio test (C_{RT}). We define:

$$C = C_{FE} + C_{RT}$$

If this were running in realtime, or in other words, the steady-state case, then the best way to look at it is to consider the complexity per comparison. For feature

extraction:

$$C_{FE} = F_C\left(S_f\right)$$

where F_C is the computational complexity of the chosen feature, many of which can be seen in Table 3.1. The second part of complexity is:

$$C_{RT} = \left(\frac{2 \cdot F_S \cdot S_W}{S_f}\right) \cdot D^2 + D^3 \tag{3.3}$$

where F_S is the sampling frequency (samples/sec), S_W is the window size (in units of seconds), and S_f is the frame size (samples/frame). The quantity $\frac{2 \cdot F_S \cdot S_W}{S_f}$ is twice the number of frames per window. Multiplying by two is necessary because in each comparison the window is essentially processed twice by looking at the left and right panes and the whole window. The square and cube powers in the equation for C_{RT} are there because of the calculation of the covariance matrix and the inverse of the covariance matrix in the multivariate Gaussian probability distribution function. See Appendix A for more details on the derivation of this equation.

3.4 Experiments

This experiment used a SparkFun 6-DOF IMU v3, seen in Figure 3.3. The SparkFun device has a Freescale MMA7260Q 3-axis accelerometer as well as gyroscopes and magnetometers, though only the accelerometers were used in this experiment. This device uses a LPC2138 ARM7 microcontroller and Bluetooth to communicate with a computer. The SparkFun IMU was rigidly mounted to a subject's right wrist while they performed sequence of activities. The device was mounted such that the accelerometer's x-axis was parallel to the forearm, pointing toward the elbow, the y-axis perpendicular to the forearm, pointing in the same direction as the thumb when it is outstretched, and the z-axis pointing into the hand from the back of the hand to the palm. All data was sampled at 100Hz and processed off-line using Matlab.

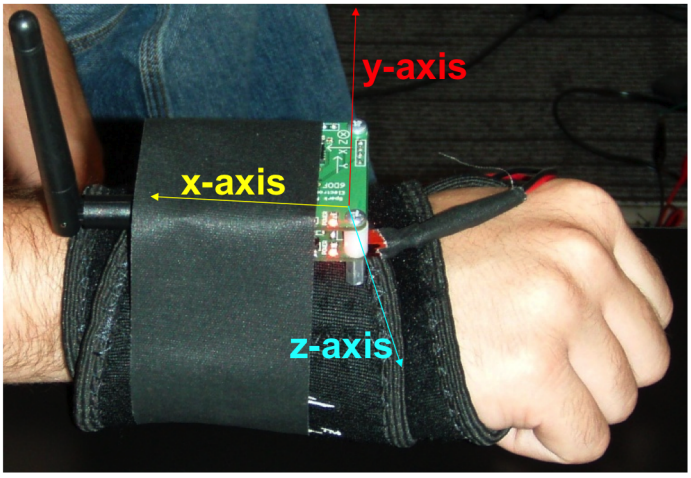

Figure 3.3: The SparkFun 6-DOF IMU v3 features a 3-axis accelerometer, 3-axis gyroscope, 2-axis magnetometer and Bluetooth connectivity. In our tests, this device was attached to the right wrist, with the x-axis parallel to the forearm. The raw data was sent via Bluetooth to a computer and saved for use in Matlab simulations.

3.4.1 Energy and Accuracy Trade-offs

We estimate the accuracy and measure the performance of each sequence of activities, for each design alternative. Accuracy is measured as Reverse F-Score (RF), while performance is computed as average runtime per activity sequence. We define runtime in this way because we want to compare the runtimes across all activity sequences, which are of different durations.

Figure 3.4 plots the performance and accuracy of all design alternatives. Out of the total 4480 combinations, only 15 are Pareto-optimal, and are connected by a curve, and marked by circles on the graph. The Pareto-optimal points are also summarized in Table 3.2. Each Pareto-optimal design point represents a design alternative for which there is no better performing (in terms of processing time) design alternative for a given F-Score. These are the most interesting points in the design space. The other permutations are either slower or have a lower F-Score. Much of the following discussion will focus on what we learn from these points.

A couple of interesting observations can be made about these Pareto-optimal de-

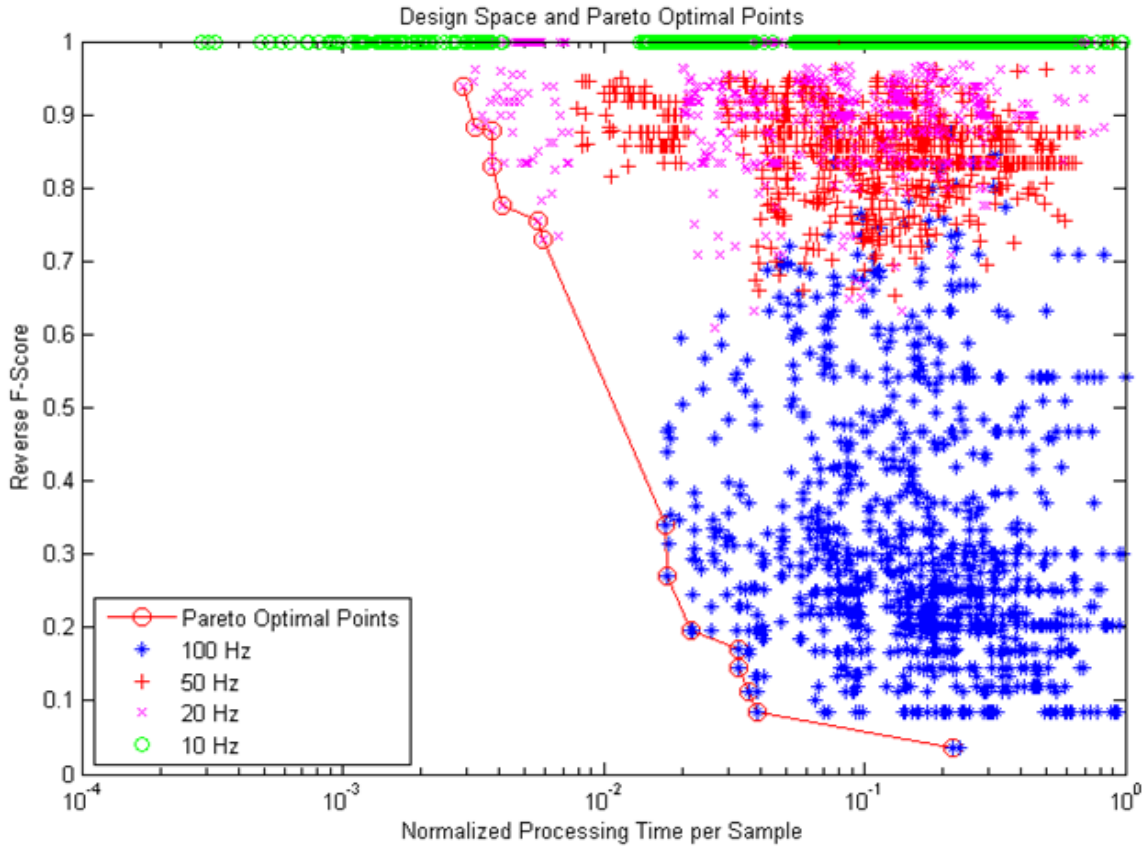

Figure 3.4: The red circles highlight the Pareto optimal points, which are dominated by the 100Hz and 20Hz sampling frequencies. All combinations with a sampling frequency of 10Hz had an RF of 1, meaning they did not detect any transitions. Note the x-axis is log scale and the y-axis is linear.

sign points. The difference in accuracy between the top two rows in Table 3.2 is only 5%, but the top combination runs approximately 5.6x longer than the second, meaning its computational complexity and energy consumption is much greater. Thus, significant energy savings can be achieved if small sacrifices in accuracy are tolerable.

The eight pareto-optimal points on the lower right side of Figure 3.4, all have a sampling frequency of 100Hz, while the seven on the top-middle are all sampled at 20Hz. Interestingly there is not much gain in accuracy when sampling at 50Hz (+ marked points) as compared to 20 Hz (points marked by x), while there is significant improvement in performance. It appears the 10Hz combinations ran very fast, but

Table 3.2: This table summarizes the system parameters of the highlighted points in Figure 3.4. Note similarities in signal, feature, frequency, and frame size.

RF	Normalized Time	Signal (axis)	Feature	Freq. (Hz)	Frame Size	Window Size (s)
0.036	0.2172	x	DCT	100	10	16
0.086	0.0388	y	min	100	20	18
0.112	0.0359	x	mean	100	20	16
0.146	0.0331	y	max	100	20	14
0.170	0.0330	x	min	100	20	14
0.196	0.0216	x	max	100	20	8
0.270	0.0176	x	min	100	20	6
0.340	0.0172	x	max	100	20	6
0.729	0.0059	x	variance	20	20	10
0.754	0.0056	x	variance	20	20	8
0.775	0.0041	x	min	20	20	10
0.829	0.0037	x	mean	20	20	8
0.878	0.0037	z	min	20	20	6
0.882	0.0032	x	mean	20	20	6
0.938	0.0029	x	max	20	20	6

simply did not have enough data to identify the transitions.

All Pareto points except the one with the lowest RF, have a frame size of 20 samples per frame. On average, the 20-samples-per-frame combinations ran faster the 10-samples-per-frame combinations, which was predicted by our model for computational complexity in the equations in Section 3.3.2. Frame size is in the denominator in Equation 3.3; therefore dividing by a larger frame size reduces the overall complexity. This also makes intuitive sense because with a larger frame size there are fewer frames, or observations, per window and features are calculated per frame.

Note also that 12 of the 15 Pareto points use the x-axis, which represents the line parallel to the forearm, from wrist to elbow. The fact that so many of the Pareto optimal points use this axis indicates that it is important for detecting the kinds of activities and transitions between the activities we tested.

3.4.2 Low-Dimension Feature Detection is better

Another interesting conclusion we can draw from the Pareto points is the fact that most of them, all except the one with the lowest RF, use simple features such as mean, minimum, maximum, and variance. These features performed very fast compared to the more complex features such as DCT, FFT, and the wavelet transformations. Even though both the wavelet transformations are O(N), just like the simple features, the wavelets are represented by a vector of coefficients, rather than a scalar, which the simple features use. This is a key difference in the runtime between the two groups.

Figure 3.5 shows the Pareto optimal points for each feature and the best overall. The best overall curve, shown in red, is the same as the curve shown in Figure 3.4. Notice how the simple feature group and the more complex feature group have similar curves within their group. The increase in runtime in the complex feature group is attributable to the increased feature dimension. Equation 3.3 shows that computational complexity is proportional to the square and cube of the dimension of the feature. These two elements dominate the equation when the dimension of the feature is high.

Figure 3.6 shows how the number of frames per window and the dimension of the feature affect computational complexity. It shows the graph of $N_f \cdot D^2 + D^3$, where N_f is the number of frames per window and D is the feature dimension. In our experiments, the feature dimension ranges in size from 1 to 20, and the number of frames per window ranges from 3 to 200. Low feature dimensions have little effect on

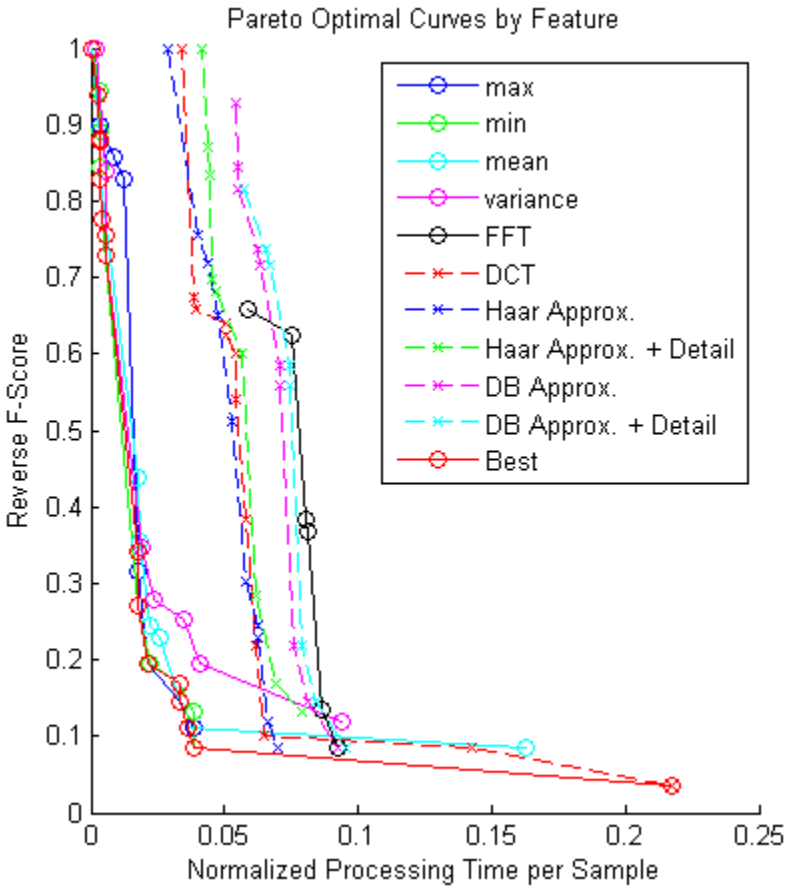

Figure 3.5: The Pareto optimal points for each feature are shown. This figure shows how feature computational complexity affects system runtime. FFT, DCT, and the wavelet approximations are vectors, but max, min, mean and variance are scalars.

complexity as the number of frames per window increases. However, a high feature dimension has a significant impact on computational complexity as the number of frames per window increases.

3.5 Summary

This chapter introduced the design space of transition detection. This design space includes the choice of activities to monitor, the sensor to use, temporal parameters such as sampling frequency and window size, and the choice of what features to calculate from the sensor signals. We began exploring how each choice in the design

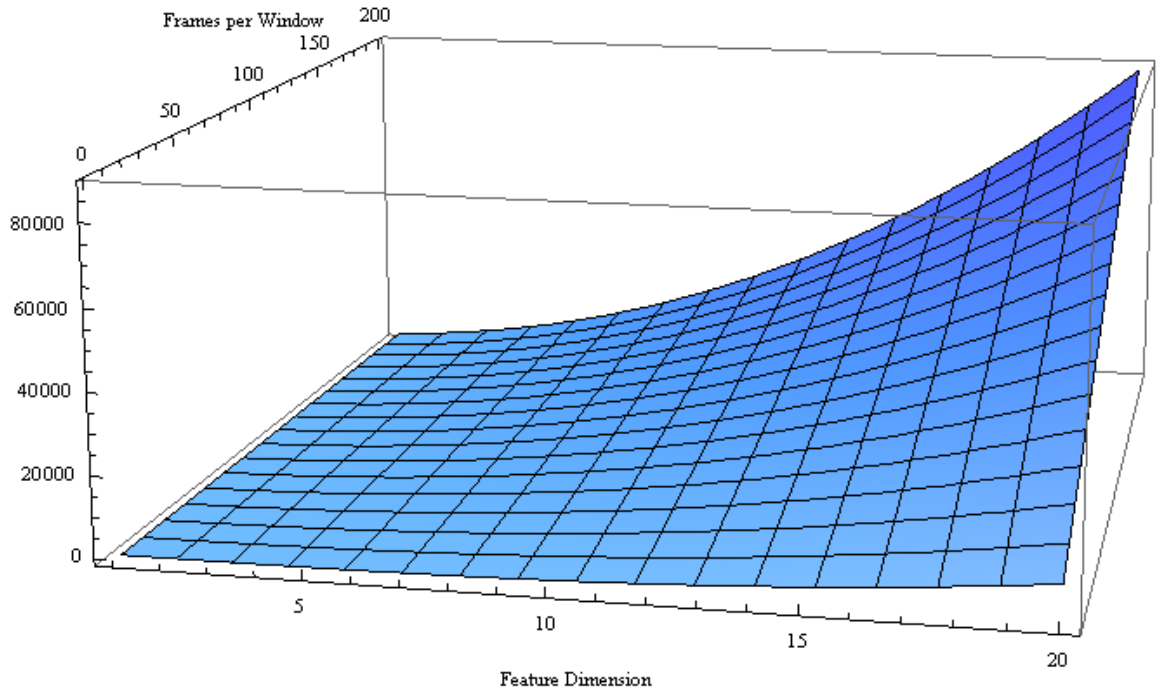

Figure 3.6: Complexity increases sharply with the number of frames per window when feature dimension in high, but is relatively flat when feature dimension is low.

space affects accuracy and energy consumption. We introduced the log-likelihood ratio test as a simple instrument of detecting transitions. Our experiments with accelerometer data from a wrist-worn device show there great variety in the design space surrounding transition detection and yielded the surprising result that simple scalar features are sufficient in many cases to detect transitions. Most importantly, our analysis shows that it is possible to achieve greater energy efficiency without sacrificing much accuracy. Our model is rather simplistic, however, in that it doesn't take into account that computing systems can enter a low-power sleep state in which no calculations are done. The next chapter expands the design space to include sleep schedules and how they affect accuracy and energy efficiency.

Chapter 4

THE EFFECTS OF SLEEP ON ENERGY/ACCURACY TRADE-OFFS

4.1 Introduction

This chapter expands the discussion on energy efficient activity recognition by extending the design space of the previous chapter to include the effects of low-power sleep states and discussion of energy efficiency choices in each stage of the activity recognition process.

As seen in the top loop of Figure 4.1, these steps each consume different amounts of energy: E_P for sleep, E_S for sensing, $E_{f,c}$ for feature calculations, and E_C for activity classification. In addition, these steps usually occur in a simple loop with a fixed sleep time and no probability of changing the order. In order to provide maximum freedom of movement for the user, and achieve maximum, uninterrupted monitoring, all this computation must be performed on a battery operated mobile device, which the user has to carry all the time. Given the limited storage capacity and the critical need to minimize the battery weight to carry, it is desirable to implement this activity monitoring system in a energy-efficient manner.

This chapter explores the energy and accuracy tradeoffs in the design of a human activity detection system. There are three main contributions:

- We explore the energy requirements at each stage of the activity recognition process, including sensing, sleep strategies, feature calculations and activity classification. Each step has its own energy requirements and decisions at each step can influence energy and accuracy measurements in subsequent steps.

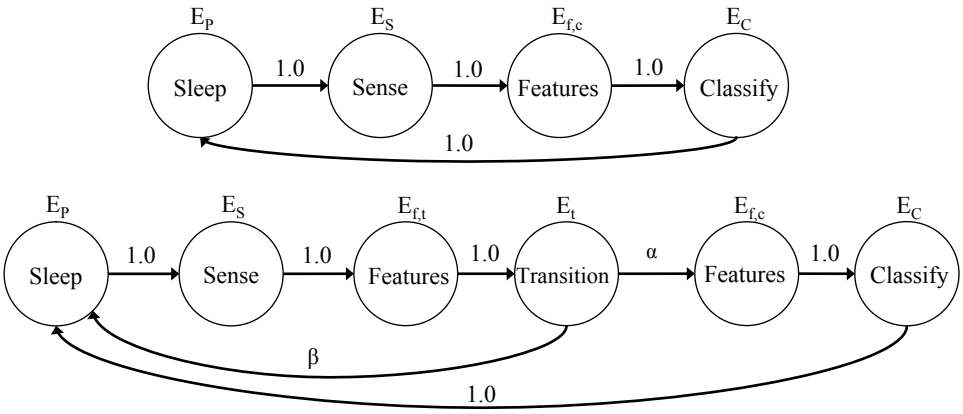

Figure 4.1: A typical activity recognition embedded system, as seen in the top loop, sleeps, senses, calculates features, classifies and then repeats. Each step consumes a different amount of energy, defined as E_P, E_S, $E_{f,c}$, and E_C, respectively, with the guaranteed probability of transitioning from one step to the next. As $E_{f,c}$ and E_C can be especially high, we propose using activity transition detection as a preprocessing step to prevent calls to energy-consuming classifiers, as seen in the lower loop. A transition detector consumes energy to calculate some feature(s), $E_{f,t}$ and to test for a transition, E_t. The probability of calculating the activity classifier features and classifier is $\alpha = P_T P_C + P_{NT} P_{FA}$ and depends on the probability of activity transition and no transition in the activity sequence (P_T and P_{NT} respectively) and the probability of the transition detector correctly identifying a transition (P_C) and raising a false alarm (P_{FA}). The probability of moving to the sleep state after transition detection is $\beta = P_T P_M + P_{NT} P_{CR}$ and depends on the probability of the transition detector missing a transition (P_M) and the probability of correctly identifying no transition (P_{CR}), in addition to P_T and P_{NT}.

- We introduce three methods of transition detection to lower the energy drain on the system, as outlined in the bottom loop of Figure 4.1. As opposed to trying to classify sensor output signals into human activity at each moment, we detect the change in the pattern of the signal rather than a change in the signal itself. This scheme is called Activity Transition Detection, and has been shown to be more energy-efficient [Boyd *et al.*, 2010]. The transition detection methods we explore include a log-likelihood test, a trained Support Vector Machine, and using simple threshold cut-off values of mean and variance. We also explore the question of when it is beneficial to use transition detection, given characteristics of the activities you want to detect (such as average activity length and prob-

ability of changing activity), and the characteristics of the transition detection methods (such as hit, miss, and false alarm probabilities).

- While some of the key design parameters, e.g., sampling frequency have been explored by previous researchers [Krause *et al.*, 2005], previous works have not performed a multi-parameter exploration that we present in this work. Our experimental results underscore the importance of design space exploration for designing an accurate, yet energy-efficient activity transition detection system. We can improve the energy efficiency of our system by more than 6× by carefully selecting design parameters and algorithms, and giving a 7% leeway on accuracy.

Section 4.2 explores the potential power savings at each step of the activity recognition process, including a discussion on some transition detection methods in Section 4.2.4. We present our experimental results in Section 4.3. We conclude this chapter with a discussion on when it makes sense to use transition detection or why it sometimes makes sense to use some computation upfront to prevent more computation downstream.

4.2 Energy Considerations in the Activity Recognition Process

From the choice of sensors, to sleep strategies, feature calculations, and activity classifiers, there are energy consequences at each step of the activity recognition process. These energy consequences usually trade increased recognition accuracy for increased energy, but hidden within the many combinations of parameters there may be some that increase recognition accuracy and decrease energy at the same time [Bharatula *et al.*, 2005]. Finding these combinations can be a difficult task; they may be revealed only after experimentation or simulation, but it is the very existence of these possibilities that should motivate researchers to consider carefully

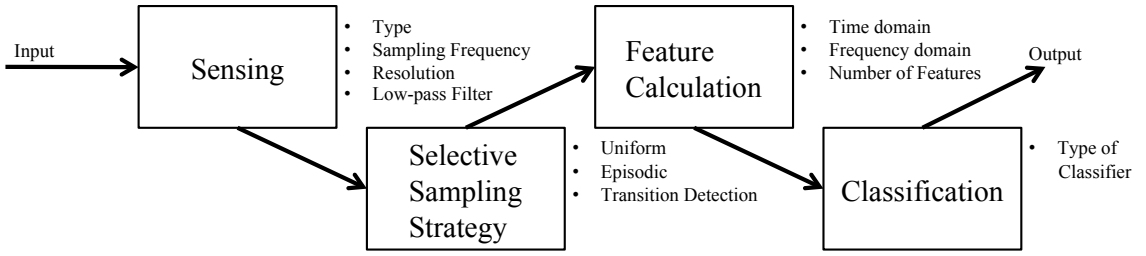

Figure 4.2: In general, there are four stages in the activity recognition process and there are energy consequences in each. Sensing includes the sensor type, sampling frequency, how many bits will be used to represent it, and any low-pass filtering. By selective sampling strategy, we refer to the pattern of activity and sleep, whether activity be continuous, interjected with regularly repeating cycles of sleep, or involve changing durations of sleep. We also include transition detection in this stage. Feature calculation refers to the choice of type and number of features to use for both transition detection and classification. In the classification stage, the number and type of classifiers used make a difference in energy consumption.

what design choices they make. In this section we will discuss the energy consequences of traditional techniques in each stage of the activity recognition process as well as introduce some new techniques. These steps are outlined in Figure 4.2.

4.2.1 Sensors

The activities to be recognized should guide the choice of sensors. A low-power activity recognition system should use the minimum number and correct type of sensors for a given task. Determining the minimum number of sensors and where to place them are interesting topics beyond the scope of this thesis. The main theme of this section is that reducing the volume of data that enters the pipeline is an important method of reducing energy. The first place to start is with the sensors.

Sensing Frequency and Resolution

A simple method to reduce the volume of data is to reduce the sampling frequency of the sensors with entering a sleep state. The sampling frequency depends on the activities to be recognized and, in order to fulfill Nyquist's theorem, should be sampled

at twice the frequency of the original signal. For sensing human movements this doesn't have to be very fast, at least compared to the capabilities of modern sensors, so sampling frequencies from 2-40Hz have yielded good results in the published literature [Stäger *et al.*, 2007; Krause *et al.*, 2005].

Another simple way to reduce the volume of data is the number of bits, or resolution, used to represent the signal from the sensor. A higher resolution analog-to-digital converter will be more sensitive to small changes in a signal, so the choice of what resolution to use depends on the coarseness of the movements. Measuring the small tremors in the hand of a Parkinson's patient requires a much higher resolution than measuring the large movements of a game controller or mobile phone. Using the lowest resolution ADC that can adequately differentiate between the activities in question will conserve energy. It does so mainly by reducing the amount of storage needed to save the signal and the amount of data to be transmitted wirelessly.

Powering off Sensors and Pre-processing

The latency of powering on sensors, after they've been powered off, needs to be considered if the system is designed to turn a sensor on and off. The resistance and capacitance of the sensor's electrical circuit determine latency because capacitors need to be charged and electrons have to flow through resistors to get to them. This latency also creates an upper bound on the sampling rate of the sensor, but this should not be a concern if one is using the minimum sampling frequency as outlined in the previous section. The sensor needs to conserve enough energy while it is off to justify the time and energy it takes to power it on again. Also, any unused sensors should be powered off if possible.

Pre-processing data adds some complexity to the system, but can be valuable in increasing the signal-to-noise ratio and accuracy of the system. It does this by

reducing high-frequency noise inherent in sensor signals. Methods to reduce high-frequency noise include low-pass filters such as a moving average, Laplacian, and Gaussian filters [Avci *et al.*, 2010].

4.2.2 Selective Sleep Strategies

Selective sleep strategies are the methods used to determine when a system activates its sensors and, perhaps optionally, classifies. As an example, the most basic and simple selective sleep strategy is continuous sampling in which a system does not sleep but always has its sensors on and classifies at fixed intervals. A more sophisticated selective sleep strategy might use knowledge about the activities to be recognized or information from the data it is collecting to lay out a schedule of when to sleep and when to wake up. There are real energy-saving possibilities by adding this layer of intelligence to an activity recognition system because of the possibility of staying in a low-power sleep mode as long as possible and only waking up when necessary, or at least when it's most likely that the system needs to be awake. Selective sleep strategies can be roughly divided into either time-driven or data-driven, which describe whether the choice of when to classify is determined by time or the content of the data being collected.

Selective sleep strategies are really a cycle of active and sleep system states. The strategies are differentiated by the duration of the sleep state and what happens during the active state. The duration of the sleep state can be static or dynamic, meaning it can be fixed and never change or change based on the data the system collects. Sleep strategies are also defined by how the active state uses what the system senses to determine whether or not to call a classifier and how to change the duration of the next sleep state. The following sections describe options for these defining characteristics found in the literature and some we introduce here.

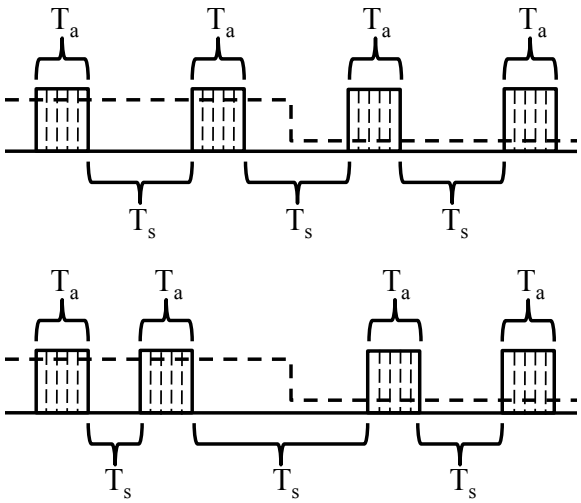

Figure 4.3: Time is divided into sequences of sleep, T_s, and analysis T_a. The top figure shows uniform sampling in which T_s is fixed. The bottom figure shows Episodic Sampling, in which T_s is variable. It increases when no change in activity is detected and decreases when change is detected. The horizontal dashed line represents the transition from one activity to another. The vertical dashed lines inside active time represent the sampling frequency of the sensors.

Uniform Sleep Strategy

Using time as the basis for when to classify is the simplest selective sleep strategy. The duration of the sleep state is fixed and chosen beforehand. A fixed sleep duration will waste energy sensing and classifying when it doesn't need to (when an activity doesn't change for a long time) and potentially miss a transition to a new activity (when the transition occurs right after it goes to sleep). In this paper we call this the uniform sleep strategy because spacing of the active states is uniformly distributed. See the top figure in Figure 4.3.

Uniform sampling is the standard to which other sleep strategies are compared and most activity recognition systems use this strategy. Most work on pattern recognition has primarily focused on recognition accuracy, not energy savings. Therefore, heuristics reducing the sampling rate have sufficed. An alternative to uniform sampling is one that changes the duration of sleep time.

Data-driven Sleep Strategy

A method called Episodic Sampling [Au *et al.*, 2009] is based on the additive increase, multiplicative decrease principle. In Episodic Sampling, the results of a classification are compared to the previous classification and if the results are the same, then some amount of time, t_{incr} is added to the duration of the next sleep cycle, up to some maximum, t_{max}. The amount t_{incr} could be fixed or a random variable chosen uniformly from an interval $[t_{incr,min}, t_{incr,max}]$. If the results of the classification comparison are different, then the duration of sleep time is decreased by multiplying it by some α, where $0 \leq \alpha \leq 1$. See the bottom figure in Figure 4.3. This allows the system's sleep time to grow gradually and conserve energy, but change sharply if needed.

Episodic Sampling is similar to the exponential back-off protocol found in the Ethernet standard [Metcalfe and Boggs, 1976]. When a collision occurs in the Ethernet protocol, a packet is delayed for some random amount of time that depends on the number of retries. The protocol essentially uses randomness to combat packet collisions. Episodic sampling, in a similar matter, uses randomness to conserve energy during the seemingly random transitions between activities.

One extension of Episodic Sampling is to use transition detection techniques to call the classifier only when a change in activity is detected. We call this method Hybrid Episodic Sampling and pseudocode for it is in Algorithm 1. Episodic Sampling differs from Hybrid Episodic Sampling in that it calls the classifier directly on line 5, instead of a transition detector. The Hybrid Episodic Sampling method combines the energy savings of being able to increase the sleep time if activities do not change frequently with the energy savings of only calling the activity classification algorithm when necessary. We discuss some transition detection methods in more detail in Section 4.2.4.

Algorithm 1: This is the Hybrid Episodic Sampling algorithm, adapted from the Au et al.'s Episodic Sampling algorithm. The amount of time the system sleeps changes according to an additive-increase, multiplicative-decrease strategy. When no change in activity detected by transition detector TD, the sleep time is increased by t_{incr}, a random variable chosen from the uniform distribution, $t_{incr} \sim \mathcal{U}(t_{incr,min}, t_{incr,max})$. If TD detects a transition, then the activity classifier C is called, and the sleep time is reduced by multiplying the sleep time by α, where $0 \leq \alpha \leq 1$.

```
1  t_sleep = 0
2  while in each episode do
3      Collect Sensor Data
4      Extract Features, F⃗ = (F_1, F_2, ...)
5      state ⇐ TD(F⃗)
6      if state has changed since last iteration then
7          activity ⇐ C(F⃗)
8          t_sleep ⇐ α · t_sleep, where 0 ≤ α ≤ 1
9      else
10         t_incr ⇐ U(t_incr,min, t_incr,max)
11         t_sleep ⇐ t_sleep + t_incr
12         if t_sleep > t_sleep,max then
13             t_sleep ⇐ t_sleep,max
14         end
15     end
16 end
```

4.2.3 Features

Features can have an impact on the energy consumption of a system in two ways. First is in the computational complexity of calculating them and second in the amount of memory required to store them. The second factor also impacts the transition detection and classification algorithms further down in the system.

Feature extraction refers to calculating some measurement on input data. Features should faithfully represent the original data and be able to distinguish between different types of activities. Researchers have experimented with a wide variety of features for different types of activity detection. Table 3.1 in Chapter 3 summarizes some of the more popular features researchers have used for inertial sensors.

4.2.4 Transition Detection Methods

If we assume that classification is a computationally expensive operation and that the uniform sleep strategy wastes energy because it will frequently classify when it is unnecessary to do so, one possible improvement is to add a function that tests if it is even necessary to classify. In other words, we wish to add a function that can accurately and in a computationally efficient manner detect whether a transition has occurred from the last cycle and then only call the classifier if a transition has occurred. Such a function would act as a gatekeeper to the classification function; it attempts to spend a little energy in computation in order to save more energy by calling a classifier less frequently.

In the next three sections we introduce methods of analyzing the data to detect transitions in real time. The first method, the log-likelihood ratio test, finds transitions by calculating the probability of the data given mean and covariance of the features of opposing sides of a window. The second method uses a trained Support Vector Machine (SVM) to detect transitions and the third uses only mean and variance thresholds.

Each transition detection method is characterized by three key numbers: E_t, P_C, and P_{FA}. E_t is the energy of the transition detector or, in other words, the time it takes the transition detector to make its decision, P_C is the probability of the transition detector correctly identifying a transition, and P_{FA} is the probability of raising a false alarm. False alarms add to energy costs because they will incorrectly trigger the classifier. The probability of missing a transition, P_M is $1 - P_C$ and will increase error. Once we have established these three methods of detecting transitions, we can move beyond the polling-loop style selective sleep strategy and into data-driven strategies that change the sleep duration based on the data. The time complexities

Table 4.1: In this table N is the number of observations, or frames, as we call them. For LLRT, D is the dimensionality of the feature vector. For SVM-transition, V is the number of support vectors.

Transition Detection Method	Complexity
LLRT	$O(D^3) + O(D^2N)$
SVM-transition	$O(NV)$
μ, σ^2	$O(N)$

of the three strategies are summarized in Table 4.1.

Log-likelihood Ratio Test (LLRT) We introduced the ideas of the Log-likelihood Ratio Test in the previous chapter, Section 3.2. That chapter demonstrated that it can be effective in detecting transitions, and that it's parameters create a rich design space. We also showed that the computational complexity of LLRT grows quickly with the dimension of the feature vector. As a refresher, LLRT is a measure of the similarity of features from two blocks of time. The previous chapter assumed these two blocks of time are adjacent to each other, but this chapter takes into account the effects of sleep and no such assumption is made. The value of LLRT will be close to 1 when the signals in two blocks of time are similar and no transition is detected and it will increase as the probability of a transition rises.

SVM-transition Detection A Support Vector Machine (SVM) [Cristianini and Shawe-Taylor, 2010] can be trained to detect transitions. Similar to the log-likelihood ratio test, two sides of a window are compared by finding the difference in feature values from several frames on opposite sides of the window as seen in Figure 4.4. The differences are the observations that become the input into the SVM.

For each window, several differences are calculated as input into the SVM, which outputs either 1 for a transition or 0 for none. If the percentage of frames in a window

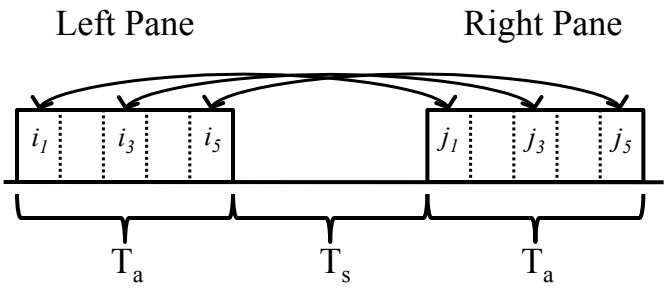

Figure 4.4: In SVM-transition detection, two window "panes" each of length T_a are separated by some sleep time, T_s. Each pane is pane is divided into separate observations, which we call frames. In this example, the first, third, and fifth frames of each pane are compared to each other by finding the per-feature difference. This difference vector is the input to the SVM-transition detector.

detect a transition reaches some threshold, it is assumed a transition has occurred, and the classifier is called. Training a SVM is computationally demanding; evaluating one is not, since evaluation is essentially an inner product of the observations with the support vectors. If we assume there are N observations and V support vectors, the complexity of SVM-transition detection is $O(NV)$.

Mean and Variance (μ, σ^2) A third method is to use the simple statistical measures of mean and variance, calculated over the entire window. These are compared to the mean and variance of the window from the previous time it was awake. If the absolute value of the difference between the current and previous window of either mean or variance is above some threshold, a transition is assumed to have occurred and the classifier is called.

This transition detection method is the simplest of the three we've discussed. Its complexity is $O(N)$ and only depends on the number of observations used in the window. Experimentation is needed to determine appropriate threshold levels. Although calculating mean and variance on the raw data is used for the experiments in this paper, they could be used on other calculated features.

4.2.5 Classifiers

A classifier is traditionally judged on its recognition accuracy. However, in low-power human activity analysis, the computational complexity of determining the classifier outcome is critical. There are many different types of activity classifiers that vary greatly in the complexity required to train and test them. Here we discuss three common classifiers and how their complexity may influence a low-power device. We discuss k-nearest neighbor, decision trees, and support vector machines.

K-Nearest Neighbors (KNN)

K-nearest neighbor classifiers are widely used in pattern analysis as baseline, non-parametric classifiers. They work by measuring the distance of new observations to the stored training data. Distances are defined by standard Euclidian distance, which can be weighted or not. New observations are classified by taking the classification of the majority of its k-nearest neighbors.

KNN can be quite computationally complex because the distance must be calculated between each new observation and all the training data. The computational complexity of the KNN classifier is $O(ND)$, where N is the number of training examples and D is the dimensionality of the feature vector. We can see that KNN can quickly become too computationally complex to run in real time for problems with a large number of training examples or features.

Decision Tree

Decision trees are another baseline classifier used in activity recognition. They use information gain to create a hierarchy of decision nodes to classify activities. Decision trees are well suited for problems in which observations are represented by attribute-

value pairs and the output has discrete values. They are also robust to errors and ambiguities in training data [Mitchell, 1997]. There are also several variations of decision trees, such as decision stumps (one-level decision trees) and random decision forests, an ensemble technique that combines the output of many decision trees each trained on a random subset of features.

The complexity of a decision tree depends on the number of nodes and how they are organized in the hierarchy. In learning a decision tree, attributes that have high information gain are placed closer to the root, which gives decision trees an inductive bias that favors smaller trees. Some researchers have chosen decision trees over better performing classifiers (in terms of accuracy) because of their low computation complexity [Stäger *et al.*, 2007; Bharatula *et al.*, 2005].

Support Vector Machines

Support vector machines (SVM) are a supervised learning method that use a number of support vectors to define a decision boundary between two classes. In the most basic SVM implementation, labeled, two-class training data are mapped into a hypothesis space such that examples of the two classes are separated by a wide margin. Training samples along the margin become the support vectors that define the decision boundary. Extensions to the original SVM definition include using kernels to map the original features to higher dimension feature spaces; this is known as the "kernel trick". This gives SVMs the ability to linearly separate two classes in a higher dimension feature space that are not linearly separable in the original feature space. Two common kernel functions are the polynomial function and the Gaussian radial basis function.

As mentioned previously, SVMs are computationally intensive to train, but not to classify. This is because evaluating a SVM is essentially an inner product of the

observation with the support vectors. However, a SVM can be inefficient if it has a large number of support vectors. This can be brought on by poor class separation from poorly chosen features.

4.3 Experiments

In this section, we discuss activities we are interested in recognizing, metrics, data used for testing, experimental results including discussion of the results.

4.3.1 Activities

We are most interested in activities of daily living (ADL). An ADL is any routine activity we perform either for leisure or to take care of ourselves. A sensible and useful measure of disability is the ability to perform ADL.

In this work, we focused on sitting, standing, walking, reaching and eating as ADL. Sitting, standing, and walking were chosen to test the transition detectors ability to distinguish between low-frequency changes in posture (from sit to stand or stand to sit) and test their ability to handle high-frequency noise from walking data. Reaching and eating are similar in that they both involve large arm movement, but thy have distinguishable motion patterns. Reaching, here, is meant to be reaching for a cup or other object in front of you. Eating is the lifting of the hand from approximately the waist to the mouth. Reaching and eating are activities of daily living that therapists of stroke survivors are particularly keen on seeing in their patients.

4.3.2 Data

We used the same experimental setup as in Chapter 3 and seen in Figure 3.3. To conduct more experiments on transition types not found in the data we collected, we created our own activity sequences by copying, replicating and repositioning the

existing data. We created our own transition table to define how likely one activity was to transition to another, assuming unequal probabilities of transitioning between activities. We assumed the duration of each activity was a Gaussian variable, so we gave each activity a mean and standard deviation. We generated 100 sequences that contained 9 transitions each. The average duration of an activity before a transition is 29.7s with a standard deviation of 6.3s.

Although the data was collected at 100Hz, the data was downsampled from 100Hz to 10Hz in 10Hz intervals to simulate having a slower underlying sensing frequency. The tests were all run at the different sampling frequencies as well.

4.3.3 Design Space Exploration

There are two major aspects of the design space that are the focus of this paper: features and sleep strategies.

Features In this work, we chose the following features: minimum, maximum, mean, and variance of each of the three axis of the accelerometer, for a total of 12 features. We chose to focus on these simple scalar features because our previous work [Boyd *et al.*, 2010] showed that simple scalar features were faster and just as accurate as vector-based features such as the FFT, DCT, and wavelet coefficients. We set the frame size to be 20 samples, regardless of the underlying sampling frequency, and calculate features by frame. This has the effect that at 100Hz, one frame is $0.2s$, but at 10Hz, one frame is $2s$.

Sleep Strategies We examined several sleep strategies. We started with the basic uniform sleep strategy with fixed sleep times of 0, 1, 2, 4, 8, 15, 20, 25, 30, 35, and 40 seconds. We used the uniform sleep strategies to compare transition detection meth-

ods against no transition detection at all. Each of the transition detection methods had their own set of parameters, each affecting the energy and accuracy in some way.

- The parameters in LLRT are the signal and feature to use, how large a window of time to examine the signal (8, 16, and 32 seconds), and what threshold to use to determine what does and does not constitute a transition.
- For SVM-transition, we look at the same window sizes as LLRT.
- For mean and variance we look at different threshold values for mean and variance. We used a fixed window length of $T_a = 2.2s$ for this method.

We also experimented with Episodic Sampling, both as it was described by Au and in hybrid form, using our Mean and Variance transition detection method. The variables for Episodic Sampling include the multiplicative decreasing factor α, the maximum sleep time, t_{max}, and the interval over which incremental amounts of sleep time are chosen from, $t_{incr,max}$. For maximum sleep time, we use the same durations as the fixed sleep times we used for uniform sampling. We define $t_{incr,max}$ by the maximum amount of time the sleep increment, t_{incr}, could be chosen from. Remember, Episodic Sampling protocol adds an amount of sleep chosen uniformly at random from some interval whenever no change in activity is detected. In our experiments we used $t_{incr,max} = \{1, 2, 4, 8, 16, 32\}$ seconds. The minimum sleep increment was fixed at our lowest unit of time increment, one frame.

Activity Classification We used one classifier to classify all the activities for all our experiments, a multi-class SVM using the LIBSVM library for Matlab [Chang and Lin, 2001]. To distinguish this SVM from the SVM used for transition detection we call the former SVM-activity and the latter SVM-transition. SVM-activity was trained using an equal distribution of data from each of the five activities and all 12

Table 4.2: The confusion matrix for SVM-activity shows generally good results. The average testing accuracy is 96.4%. We used the LIBSVM library for Matlab to train a ν-SVC with a Radial Basis Function kernel. A grid search over the ν and γ parameters yielded $\nu = 0.1$ and $\gamma = 1/12$ having the best average accuracy using 10-fold cross validation. Each value is a percentage.

		Predicted				
		Sit	Stand	Walk	Reach	Eat
Actual	Sit	98.4			1.6	
	Stand	0.5	94.7	4.8		
	Walk	0.5	2.7	96.8		
	Reach	1.6			94.7	3.7
	Eat	0.5		0.5	1.6	97.4

features described above. We trained a ν-SVC with a Radial Basis Function kernel and performed a grid search over the ν and γ parameters. Testing was done using 10-fold cross validation and the results are summarized in Table 4.2. Average testing accuracy is 96.4%.

4.3.4 Evaluation Metrics

We evaluate our experiments using two broad categories: accuracy and estimated energy consumption. We measure accuracy with precision, recall, F-score, and activity sequence reconstruction error. Estimated energy consumption based on the runtime of the experiments.

Accuracy

Similar to the previous chapter, we use Precision, Recall and F-Score to evaluate the accuracy of our transition detection algorithms. These metrics are based on hits, misses and false alarms. See Section 3.3 for the definitions of these measurements.

We also measure error based on a frame-by-frame comparison with the ground truth. This could also be called reconstruction error because it is a measure of how well the sleep strategies and transition detectors reconstruct the sequence of activities in the ground truth. More formally, we define the reconstruction error, e, to be

$$e = \frac{\sum_{i=1}^{N} x_i \wedge y_i}{N}$$

where x_i is the ground-truth label for the frame, y_i is the predicted label, and N is the total number of frames in the data sequence. For this measurement, we assume a "sample-and-hold" strategy where the results of the activity classifier are held until the next time the activity classifier is called. A natural effect of this strategy is that experiments with little or no sleep time will detect transitions close to when they actually happen. Experiments with longer sleep time may potentially miss the transitions and introduce large errors.

Energy Consumption

We estimate the energy consumption and, by analogy, the computational complexity of a particular permutation of system parameters by measuring the runtime of each experiment. This empirical measurement gives us an estimation of the energy consumption of a system that would implement the aforementioned sleep strategies and transition detectors since energy consumption is proportional to runtime [Roy and Johnson, 1997; Zotos *et al.*, 2005; Tiwari *et al.*, 1996].

For our experiments, we used the *tic* and *toc* functions in Matlab to calculate the runtime of feature calculation (our sensing estimate), transition detection, and activity classification. These measurements are the basis for our energy consumption estimates. Let the energy required to sense and calculate features be E_s, to analyze and detect a transition be E_t, and to classify be E_c.

4.3.5 Uniform Sampling

Figure 4.5 summarizes the results of comparing the uniform sleep strategy for the baseline, no-transition-detection method and our three transition detection methods. The Pareto-fronts and their respective regressions are shown. Note that the y-axis is reconstruction error and the x-axis is time, so points closer to the origin represent combinations of system parameters more accurate and consume less energy. The Pareto-front represents the combinations that are most accurate for a given level of energy consumption. Points above those in the Pareto front are less accurate; points to the right are slower and use more energy. The purple line is the baseline, no transition detection, and points below and to the left of it are more accurate and consume less energy. As you can see, the μ and σ^2 transition detector, the least complex of the transition detection methods we've described, is almost always faster and more accurate than no transition detection at all. The LLRT method performed about the same as no transition detection and SVM-transition was never faster and more accurate. Figure 4.5 shows only the Pareto fronts for each transition detector. Scatterplots that show how each permutation of the design parameters performed in the error-time space can be found in Appendix B.

The Great Compromise Figure 4.5 shows that great energy savings can be obtained by not using the most accurate combination of system parameters. The most accurate combination used no transition detection and never slept; it was approximately 98% accurate in reconstructing the activity sequence. Compare this with a value near the 0.1 error mark, which uses μ, σ^2 transition detection, It is approximately 90% accurate in reconstructing the original activity sequence and runs approximately 6$\times$ faster than the most accurate combination and uses proportionally less energy. We call this the "great" compromise because of the great energy savings

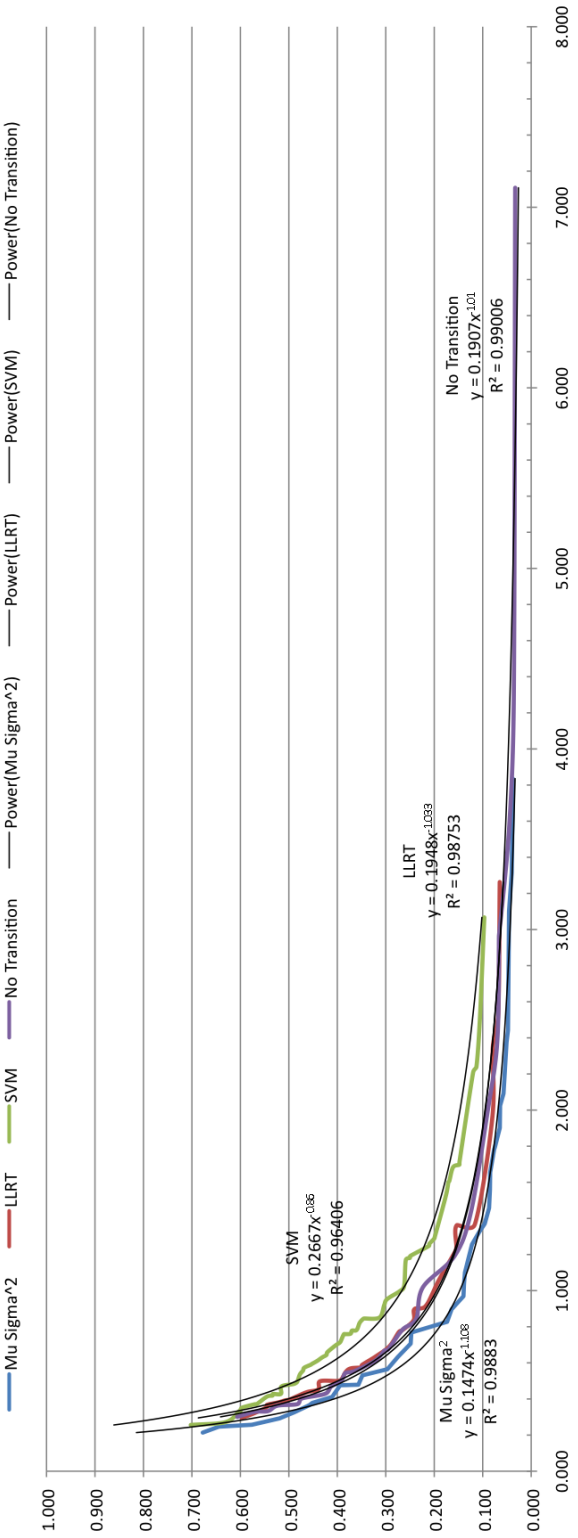

Figure 4.5: The Pareto-fronts of all transition detectors and no transition detection with their respective regression curves. Note here how close each of the four curves approach $y = \alpha/x$ with low error ($R^2 \approx 1$).

Table 4.3: Hit, miss, and false alarm values were obtained by testing the transition detectors on 1000 pairs data sequences known to be of differing activities and 1000 pairs of data sequences known to be the same activity. Please note these values represent some of the best performing (in terms of accuracy and energy consumption) combination of system parameters. Each combination will have its own hit, miss, and false alarm probabilities. Each transition detection method also consumes energy to calculate its features, $E_{f,t}$, and to determine if there is a transition, E_t. E_c is the energy our activity classifier consumes; for our experiments $E_c = 6.15 \times 10^{-4}$s. The ratio E_t/E_c is important in determining if energy can be preserved by transition detection.

Transition Detector	P_M	P_{FA}	P_C or Recall	Precision	F-Score	$E_{f,t}$ (μs)	E_t (μs)	E_t/E_c
LLRT	0.012	0.304	0.988	0.765	0.862	9.55	456	0.64
SVM-transition	0.109	0.076	0.891	0.921	0.906	14.2	144	0.39
μ, σ^2	0.009	0.237	0.991	0.807	0.890	57.3	15.6	0.10

to be had if a little drop in accuracy can be tolerated.

Table 4.3 shows some other interesting properties of the transition detection methods; it summarizes their Precision, Recall, F-score, and energy characteristics. Note that SVM-transition detection has the highest F-score on account of its low probability of false alarm, P_{FA}, relative the other two. LLRT and the μ and σ^2 transition detectors are very eager to identify transitions hence they have very low miss probability, P_M values, but high P_{FA} scores. F-score simply does not give a complete picture. It does not include energy measurements. The value of E_t for SVM-transition is an order of magnitude slower than for μ and σ^2 transition detection. Also, we see in Table 4.3 that the probability of missing a transition, P_M, for SVM-transition is an order of magnitude higher than the other two methods.

Power Balance Inequality Let us assume a uniform sleep strategy: in each cycle, the time the system sleeps is T_s and the time the system analyzes the data is T_a. Let the fraction when the system is awake be $f_a = T_a/(T_a+T_s)$. When the system is awake,

it first calls a transition detection algorithm. If the transition detector indicates that there has been a transition, then an activity classifier is invoked. Furthermore, let us assume that the transition detector makes errors, which we represent with the P_C, P_M, and P_{FA} as described in Section 3.3. The system will use energy to sense (E_s), detect a transition (E_t), and classify (E_c). Finally, we note that the two classes—transition, no transition—have different probabilities: P_T and P_{NT}, both of which depend on the fixed sleep time T_s.

Any time that we sense, we save energy in the following conditions:

$$f_a \cdot (E_s + E_c) \geq f_a \cdot (E_s + E_t) + f_a \cdot (P_T P_C + P_{NT} P_{FA}) \cdot E_c \tag{4.1a}$$

$$E_c \geq E_t + (P_T P_C + P_{NT} P_{FA}) \cdot E_c \tag{4.1b}$$

$$E_t \leq (1 - (P_T P_C + P_{NT} P_{FA})) \cdot E_c \tag{4.1c}$$

There are three interesting insights from Equation (4.1c), given specific values for E_t and E_c. First, transition detection is an asymmetric classification problem because $P_T \ll P_{NT}$. This implies that the relationship between E_t and E_c is *not very sensitive* to the exact value of the classification rate P_C, since the value of P_T is very small. Second, the critical parameter affecting the relationship is the false alarm rate P_{FA}. This outcome is intuitive. The probability that any segment is *not* a transition is high (i.e. $P_{NT} \approx 1$), and if the transition detector has many false alarms, we run the classifier many times, thus expending significant energy. Third, the inequality has a greater chance of being satisfied if $E_t \ll E_c$.

If we want to ensure that Equation 4.1c holds for highly asymmetric class prior probabilities, we need to ensure that the false alarm rate is low and that we pick a transition detector that is cheap in terms of energy. Another way to understand Equation (4.1c) is that the ratio $E_t/E_c \leq 1 - (P_T P_C + P_{NT} P_{FA})$ in order for the system to conserve energy by transition detection. This is demonstrated in Table

Table 4.4: For each transition detector, there is a point when it no longer makes sense to use transition detection. Energy can be conserved as long as $1 - (P_T P_c + P_{NT} P_{FA}) \geq E_t/E_c$, indicated by the bold values in this table. This value is dependent on the sleep time, the average duration of activities, the probability of transitions (P_T), and no transitions (P_{NT}) in the data, each transition detector's probability of correctly identifying a transition (P_c) and the probability of raising a false alarm (P_{FA}). For the data we collected, the average activity length is 29.7 seconds. The values of E_t/E_c for each transition detector are in Table 4.3.

				$1 - (P_T P_c + P_{NT} P_{FA})$		
Sleep (s)	P_T	P_{NT}	$\frac{\text{Sleep}}{\text{Avg. Activity Length}}$	μ, σ^2	LLRT	SVM-transition
0	0.07	0.93	0.00	**0.71**	**0.65**	**0.87**
1	0.10	0.90	0.03	**0.69**	0.63	**0.85**
2	0.13	0.87	0.07	**0.67**	0.61	**0.82**
4	0.19	0.81	0.13	**0.62**	0.57	**0.77**
8	0.31	0.69	0.27	**0.53**	0.48	**0.67**
15	0.53	0.47	0.51	**0.36**	0.33	**0.49**
20	0.68	0.32	0.67	**0.25**	0.23	0.37
25	0.80	0.20	0.84	**0.16**	0.15	0.27
30	0.89	0.11	1.01	0.09	0.09	0.20
35	0.93	0.07	1.18	0.06	0.06	0.17
40	0.94	0.06	1.35	0.05	0.05	0.16

4.4 and visualized in Figure 4.6, which show that there is a certain point for each transition detector where there is no longer a benefit of doing transition detection.

4.3.6 Data-driven Sampling

We now move to experiments and discussion of using data analysis to change the sleep strategy. In the case of episodic sampling, we use the results of an activity classifier at two time instances to change the amount of sleep. Sleep time additively increases when no change in activity is detected, it multiplicatively decreases when

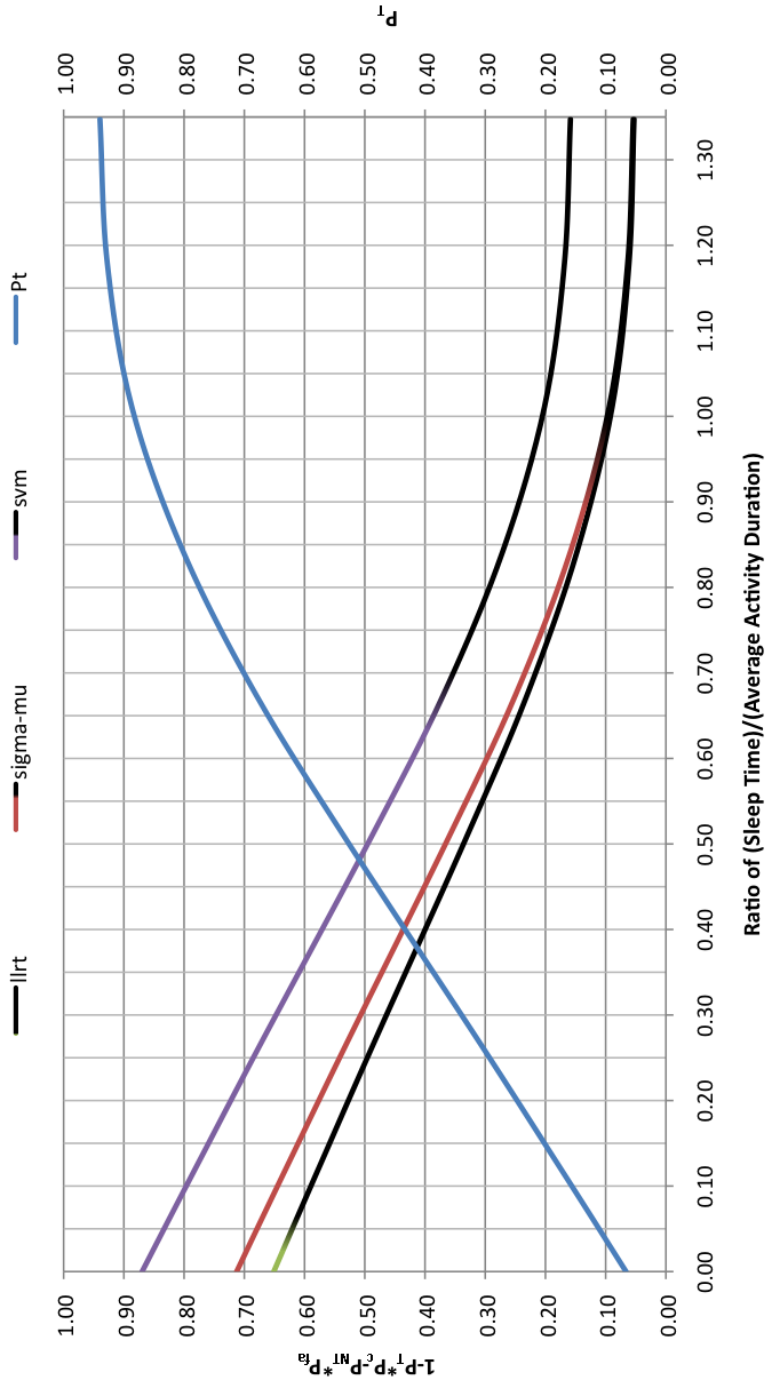

Figure 4.6: This figure visualizes the data in Table 4.4. The values of $1 - (P_T P_c + P_{NT} P_{FA})$ for each of the three transition detection methods decrease linearly as the probability of transition, P_T, increases. Energy is conserved in the colored regions of the lines. The long extention of the colored region of the μ, σ^2 line indicates it would be able to conserve more energy than the other two methods because it is still effective when sleep times and the probability of transitions occurring are high.

a change is detected. In hybrid episodic sampling, a transition detector is used to determine if a call to the activity classifier is needed. Our experiments focused on the effects of the multiplying constant α, $t_{incr,max}$, and t_{max} on energy and accuracy.

For both types of data-driven sampling, the choice of α seems to have little overall affect on the energy consumption and accuracy of the system. Values for α of 0.1 and 0.9 will aggressively or slightly reduce the sleep time, respectively, when a transition or new activity is detected, so one would expect $\alpha = 0.1$ to use much more energy than $\alpha = 0.9$. In our experiments, for a given max sleep time, t_{max}, and max sleep increment, $t_{incr,max}$, values of $\alpha = \{0.1, 0.3, 0.5, 0.7, 0.9\}$ form a small cluster, without a significant difference between energy and accuracy of $\alpha = 0.1$ and $\alpha = 0.9$. For the remainder of our data-driven sampling experiments, we used $\alpha = 0.5$.

The values of $t_{incr,max}$ and t_{max}, however, do have a big impact on energy and accuracy. Figure 4.7 shows a comparison between episodic sampling, uniform sampling with no transition detection, and hybrid episodic sampling using μ, σ^2 as the transition detector. For our experiments we used values of $t_{incr,max} = \{1, 2, 4, 8, 16, 32\}$ and $t_{max} = \{0, 1, 2, 4, 8, 15, 20, 25, 30, 35, 40\}$ in units of seconds. As our data-driven sampling algorithms depend on random numbers, each experiment was run 50 times and their scores averaged. Figure 4.7 gives us several insights into how $t_{incr,max}$ and t_{max} affect energy and accuracy, and how they relate to the uniform, no-transition detection strategy.

The first is that for a given maximum sleep time for both uniform and episodic sampling, uniform sampling is faster and uses less energy, but almost always has a marginally higher error rate. Episodic sampling is more accurate than uniform sampling when $t_{incr,max}$ is small, but it is slower and will consume more energy. There is no clear advantage, in terms of energy and accuracy, to use episodic sampling over uniform, no-transition detection when they have similar maximum sleep times.

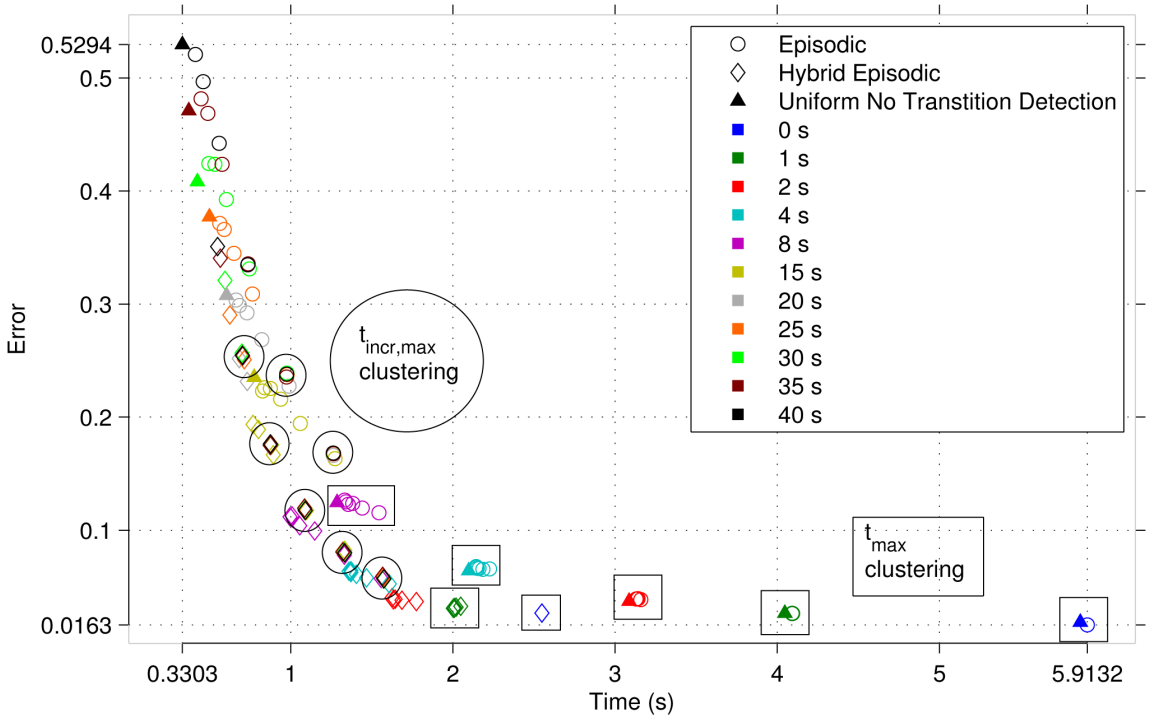

Figure 4.7: Here we compare how changing maximum sleep time (t_{max}) affects episodic sampling (circles), uniform sampling with no transition detection (triangles), and hybrid episodic sampling (diamonds) at a sampling rate of 100Hz. Each color represents a different t_{max}. Both the episodic and hybrid episodic sampling strategies used the same values for $t_{incr,max}$. There are three important things to note. First, the uniform, no-transition detection strategy is always faster than episodic sampling for a given t_{max}, but it usually has the highest error. Second is that including transition detection in the hybrid episodic sampling algorithm results in faster, lower power, and more accurate results in the region closest to the origin, representing the best trade-off between energy and accuracy. Third is that there are two types of clustering, one around small values of t_{max} for all values of $t_{incr,max}$, and another for small values of $t_{incr,max}$ and large values of t_{max}.

Hybrid episodic sampling does offer a clear advantage over uniform, no-transition detection sampling for a given t_{max}, in terms of both energy and accuracy. In Figure 4.7, in the region closest to the origin, which represents the best trade-off between energy and accuracy, the hybrid episodic strategy is clearly faster and more accurate than the other two strategies. Again we see a situation similar to the Great Compromise described above where a marginal reduction in accuracy can be several times faster and consume much less energy than the most accurate strategies.

Another insight Figure 4.7 gives us is that there appears to be two types of clustering based on t_{max} and $t_{incr,max}$. The first is obvious to explain. When t_{max} is small, it doesn't matter how big $t_{incr,max}$ is because $t_{incr,max}$ reaches t_{max} and stays there, basically reducing to uniform sampling. This is why they cluster close to the uniform sampling strategy. The second type of clustering occurs when t_{max} is large and $t_{incr,max}$ is small. When t_{max} is large, there is greater diversity of energy and accuracy values based on $t_{incr,max}$. Clustering occurs around the different values of $t_{incr,max}$, suggesting that t_{incr} never reaches t_{max} and each value of $t_{incr,max}$ in these cases produces similar strategies. In terms of finding a good trade-off between energy and accuracy, it appears t_{max} gets you in the general area and values of $t_{incr,max}$ are for fine-tuning.

4.4 Discussion

In exploring the stages of the activity recognition process we have not found one area that significantly improves accuracy or reduces energy consumption, but rather some rules-of-thumb that researchers should explore in their own projects. The first is that less is more, or at least good enough. We found using simple, scalar features were good enough, in terms of accuracy, as more computational complex, vector-based features and consumed less energy for our transition detection methods. Along this same vein, we observed that a relatively simple transition detection method using only the mean and variance of simple features was more accurate and consumed less energy than more complex log-likelihood ratio test and SVM transition detectors. Researchers should explore ways to simplify computations and reduce data storage in their projects. An analogy comes to mind: when it comes to energy consumption, data are like the calories we consume, the more there are, the more energy the system consumes to process them all.

Another rule-of-thumb is to explore the idea of using some resources to prevent using even more resources later on. This is essentially what transition detection asserts and what our experiments confirm. We wanted to see if using a relatively simple process could be effective as a type of gate-keeper to a more complex process, calling on the more complicated process only when needed. We found confirmation that this works in both fixed-interval (uniform) and variable-interval (episodic) sleep patterns. This is a powerful idea that has implications beyond activity classifiers.

One more rule-of-thumb is that significant energy savings are possible if a marginal reduction in accuracy is tolerable. We call this the Great Compromise in this chapter. We use "great" a little tongue-in-cheek because in some applications it is not really much of a compromise at all. This has application to any battery-powered or energy-constrained systems. While we do not recommend this rule-of-thumb for applications where accuracy is of the utmost importance, we recognize that not all systems require perfect accuracy and researchers should evaluate their systems to see if some compromise in accuracy can result in reduced energy consumption.

4.5 Summary

We have presented a model for activity recognition and described the energy considerations at each stage of the model. For sensing, this includes the number and type of sensors as well as the sampling frequency, resolution and pre-processing such as a low-pass filter. At the selective sampling strategy stage, we explored using fixed-interval sleep times and variable-interval sleep times that change with the sequence of activities. We also compared three transition detection methods as a means to conserve energy: a log-likelihood ratio test, a SVM, and one simply using mean and variance. Each stage of the activity recognition process has parameters that can be tweaked in the search for a compromise between energy and accuracy.

We also note that features and classifiers affect energy consumption in the computational complexity of calculating them and in the amount of memory needed to store the results (their time and space complexities). Vector features not only take up more storage space, but they can greatly increase the time complexity when used in sophisticated downstream operations such as matrix inversion or calculation of a covariance matrix. Classifiers, too, vary in their complexity. Some classifiers, such as KNN have little up-front or offline preparation and put most of the complexity at the time of classification. Other classifiers, such as decision trees and SVMs, do a considerable amount of work up-front and offline so that evaluating them is a less computationally complex process. These latter type of classifiers are better-suited for embedded systems.

Our experimental results led us to what we call the Great Compromise, where a 7% sacrifice in the accuracy of reconstructing the activity sequence can bring a 6× increase in energy efficiency. The consequence of such energy savings in a wearable sensor would mean longer intervals between recharging or replacing batteries or smaller, more lightweight batteries, thus reducing the entire package size. In any case, a more energy efficient system leads to a better user experience.

Chapter 5

ENERGY AND DATA MODELS, PLUS TRANSMISSION EFFECTS

Sensing and wireless transmission are the two main consumers of energy in a wearable system. We propose strategies to reduce the amount of time in these two states while minimizing reduction of accuracy and usefulness of the overall system. The efficiency of modern embedded processors allows an embedded system to conserve energy by transmitting features and activity classifications instead of raw data, thus reducing the amount of data to be transmitted. We model the energy a wearable system consumes when data is transmitted at any of four stages: the raw data, features, activity stream, or activity transitions. Transmission at any stage results in its own energy-accuracy trade-off.

We recognize that there is an inherent trade-off between energy consumption and any measurement of accuracy for a given system. In general, the more energy the system consumes, the more accurate it is. However, greater energy consumption creates practical obstacles to the design and implementation of effective wearable sensors. Greater energy consumption means a larger battery is needed for continuous operation. A larger battery may limit where a sensor can be placed or become a burden and hindrance for the person wearing it. We also recognize the importance of advancements in creating energy efficient electronic hardware, namely low-power sensors and microcontrollers. However, we believe there are many ways the software that controls the hardware can be optimized to reduce energy consumption.

In Figure 5.1 we see that sensing and transmission consume the most energy per byte. Our goal is to investigate methods of reducing the time spent sensing and transmitting data without sacrificing too much of the accuracy of the system. In this

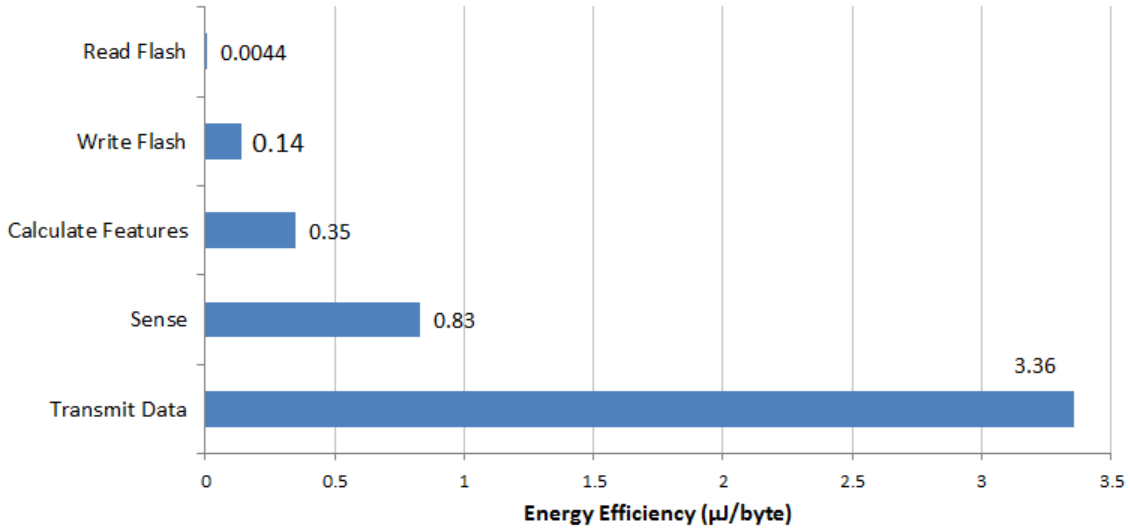

Figure 5.1: Here is shown the energy consumed per byte of data generated, processed, or transmitted by the Texas Instruments eZ430-Chronos Development Tool. The eZ430-Chronos is a programmable, wristwatch-like device that features a 3-axis accelerometer, altimeter, flash storage, and wireless transceiver. The values in this figure are drawn from the device's official specification sheet and measurements done by the author. These measurements assume data is collected at 100Hz from a 3-axis accelerometer, that mean and variance are calculated every 128 samples, and that the radio broadcasts at maximum power and capacity.

chapter we focus mainly on how to reduce the amount of data to be transmitted, since it consumes the most energy per byte and discuss briefly methods of reducing sensing. This chapter makes the following contributions:

- Energy consumption models for sensing, feature calculations, reading and writing to storage, and transmission
- Data creation models for the amount of raw data collected and amount generated during feature calculations
- Models of system-wide energy usage when transmitting raw data, features, or activity classifications

We have analyzed these models on a Texas Instruments eZ430-Chronos Development Tool, hereafter called the TI Chronos watch. Using measurements of how much

current the TI Chronos watch draws for different operating modes, we show how much energy can be saved by reducing the amount of data generated and transmitted.

In Section 5.1 we develop our models for how much energy is consumed during the stages of activity recognition and how much data is generated during sensing and feature calculations. We also model the energy consumed when data is transmitted at different stages of activity recognition. Then in Section 5.2 we summarize methods used to sense less. In Section 5.3 we present our findings with the TI Chronos watch.

5.1 Energy and Data Models

5.1.1 The Mobile Sensor

An embedded sensing system can be modeled with four stages: sensing, feature calculation or compression (which may include any preprocessing of the data), storage, and transmission. Sensing and transmission are the major consumers of energy in an embedded system.

Sensing is a major consumer of energy in an embedded system. Not only does the sensing frequency directly affect the current draw in the system (the higher the frequency, the higher the current draw), but the amount of data generated has ripple effects in the energy consumed by stages downstream in the form of increased processing time in the compression stage, and more data to be stored and/or transmitted.

Transmitters are most efficient (in terms of mW/byte) when they are used at or close to their capacity. Storing data locally in non-volatile memory such as flash consumes energy to read and write, introduces latency into the system, but is faster and uses less energy than transmitting that data wirelessly. It is more energy efficient to store data in a buffer and transmit all at once, allowing the transmitter to operate at or near its capacity, than to continuously stream the data. This assumes the

Table 5.1: Variables for Energy and Data used in this Chapter

Variable	Description
t	Time, in seconds, that a device is actively sensing in a day
F_S	Sampling Frequency (Hz)
N_C	The number of channels
N	$F_S \cdot t \cdot N_C$. This is the total number of samples produced in a day.
CO	Number of cycles to compute feature per sample (cycles/sample). For example addition or multiplication may take 5 CPU cycles.
F_O	Operating frequency of the microcontroller (Hz)
E_{feat}	Energy (mWs or mJ) to calculate features for entire day
E_{sense}	Energy (mWs or mJ) to sense for entire day
D_{raw}	Raw data generated in a day (bytes)
D_{feat}	Feature data generated (compressed) in a day (bytes)
D_{act}	Activity classification data generated (compressed) in a day (bytes)
S_{WF}	Wavelet Filter Size
S_G	Grouping Size, fastest if it's a power of 2, for FFT
Ψ_{cpu}	Power of active mode (mW)
Ψ_{sense}	Power of sensing mode (mW)
Ψ_{read}	Power of reading from flash memory (mW)
Ψ_{write}	Power of writing flash memory (mW)
t_{write}	Time to write one byte of data to flash memory (seconds)
Ψ_{trans}	Power of wirelessly transmitting data (mW)
t_{trans}	Time to wirelessly transmit one byte of data (seconds)

wireless transmission rate is much greater than the rate that data is generated.

5.1.2 The Energy Consumption and Data Creation Model

We now discuss the energy consumed by each part of the mobile sensor. For each part we describe the energy consumed and amount of data generated over some length of time, t. We list the variables used in this document and their definitions in Table 5.1 to avoid repeating variable definitions in each section.

Sense

Inertial sensors such as accelerometers, magnetometers and gyroscopes are typical measurement sensors for mobile activity monitoring. EEG, ECG and pulse oximeters are examples of sensors that measure physiological signals. The near ubiquity of smartphones has made other sensors such as GPS and Wi-Fi radios, light and proximity sensors more prevalent and useful in activity monitoring as well. The wide range of available sensors also vary greatly in the power they use.

The power of a sensor, Ψ_{sense}, is a function of the sampling frequencies, F_S, and the number of channels, N_C. $\Psi_{sense} = f(F_S, N_C)$. Ψ_{sense} is unique to each device and should be measured empirically. The energy consumed by sensing is

$$E_{sense} = \Psi_{sense} t, \tag{5.1}$$

where t is the amount of time spent sensing.

The data produced during sensing depends on the sampling frequency, F_S, the number of channels, N_C, and the number of bytes used to represent one sample. A typical setup uses anywhere between 8 and 16 bits to represent a sampling of a sensor, thus individual sensor data points are typically represented by 2 bytes. If we define N to be the total number of samples generated over some time period t, then $N = F_S \cdot t \cdot N_C$. The number of bytes of raw data, D_{raw}, generated in that time is

twice the number of samples, N:

$$D_{raw} = 2F_S \cdot t \cdot N_C = 2N. \tag{5.2}$$

Analyze

We group the sensed data stream and then each group of samples is filtered and analyzed to calculate features. The grouping size, S_G, has a great impact on the energy consumed in calculating features and in the amount of data produced. The grouping size, S_G determines how often features are calculated. The expression N/S_G is the number of groups over a given amount of data. In previous chapters we used the word 'frame' for S_G.

The energy consumed in calculating features can be estimated based on the feature's computational complexity. Typical features for accelerometers can be roughly grouped in three categories: time-domain, frequency-domain, and joint time-frequency domain. The first category includes time-domain features such as mean and variance that are typically $O(n)$. The $O(n)$ notation refers to the computational complexity or order of the function. For $O(n)$, this means the function grows linearly with the input, as seen by the example of the formula for mean: $\frac{1}{n} \cdot \sum_{i=1}^{n} x_i$. In this category there are approximately S_G operations per group and the energy used for feature calculation is:

$$E_{feat} = \Psi_{cpu} \cdot S_G \cdot \frac{N}{S_G} \cdot \frac{CO}{F_O} = \Psi_{cpu} \cdot N \cdot \frac{CO}{F_O}. \tag{5.3}$$

The ratio CO/F_O is the number of cpu cycles per elementary operation (addition, multiplication, etc. usually take multiple cpu cycles to complete and the number varies by cpu architecture) divided by the number of cpu cycles per second, which gives us the amount of time, in seconds, of each operation.

The second category includes frequency-domain, $O(n \log n)$ features such as the

Fast Fourier Transform (FFT). Here,

$$\begin{aligned} E_{feat} &= \Psi_{cpu} \cdot S_G \log S_G \cdot \frac{N}{S_G} \cdot \frac{CO}{F_O} \\ &= \Psi_{cpu} \cdot N \log S_G \cdot \frac{CO}{F_O}. \end{aligned} \tag{5.4}$$

The third category includes time-frequency features such as wavelets. The computational complexity of wavelet analysis is $O(nS_{WF})$, where S_{WF} is the size of the wavelet filter. Given S_G lengths of data, wavelet analysis needs approximately $S_G S_{WF}$ operations and the energy consumed is approximately:

$$\begin{aligned} E_{feat} &= \Psi_{cpu} \cdot S_G S_{WF} \cdot \frac{N}{S_G} \cdot \frac{CO}{F_O} \\ &= \Psi_{cpu} \cdot N \cdot S_{WF} \cdot \frac{CO}{F_O}. \end{aligned} \tag{5.5}$$

The amount of data produced during analysis by the features can either be more or less than the raw data. Scalar features such as those from the time-domain compress the raw data, representing S_G-size chunks of data with one value. Frequency-domain features such as FFT or Discrete Cosine Transform (DCT) produce the same amount of data or more. For the first category where we calculate time-domain $O(n)$ features, we have

$$D_{feat} = \frac{2F_S \cdot t \cdot N_C}{S_G} = \frac{2N}{S_G}. \tag{5.6}$$

This makes intuitive sense because as S_G increases, larger chunks of data are represented by a single number and D_{feat} decreases. The compression ratio, D_{raw}/D_{feat}, for time-domain features is $2N/\frac{2N}{S_G} = S_G$.

In the frequency domain, an FFT produces complex numbers, which use twice as many bytes. The amount of generated data is:

$$D_{feat} = \frac{4S_G \cdot F_S \cdot t \cdot N_C}{S_G} = 4N. \tag{5.7}$$

The compression ratio for FFT is $2N/4N = 1/2$. However, the DCT produces only real values so $D_{feat} = 2N$ and its compression ratio is 1. Of course an implementation

might not use all the FFT or DCT coefficients and so the compression ratio could be higher.

Wavelet analysis on a S_G-size chunk of data produces two vectors each of length $\left\lfloor \frac{S_G+S_{WF}-1}{2} \right\rfloor$. Remembering that each number is represented by two bytes, the total amount of data produced by wavelet analysis is:

$$\begin{aligned} D_{feat} &= \frac{2 \cdot 2 \cdot \left\lfloor \frac{S_G+S_{WF}-1}{2} \right\rfloor \cdot F_S \cdot t \cdot N_C}{S_G} \\ &= \frac{2 \left\lfloor S_G + S_{WF} - 1 \right\rfloor \cdot N}{S_G} \leq \frac{2N(S_G + S_{WF} - 1)}{SG} \\ &= 2N \left(1 + \frac{S_{WF} - 1}{S_G} \right). \end{aligned} \tag{5.8}$$

In general, $S_{WF} \ll S_G$, so $D_{feat} \approx 2N$ and the compression ratio is ~ 1.

The equations in this section have shown that the energy consumed and data generated by the analysis stage depend on the grouping size, S_G and the type of feature used.

Store

Mobile sensors come with volatile and non-volatile memory. Non-volatile memory, such as flash memory, has different energy consumption for reads and writes. If we assume the device reads 2 bytes (1 word) in CO_{word} cycles, then:

$$E_{read,perbyte} = \Psi_{read} \cdot \frac{CO_{word}}{2F_O}. \tag{5.9}$$

The maximum amount of time it takes to write a byte to flash memory is usually a constant in the device specifications.

$$E_{write,perbyte} = \Psi_{write} t_{write} \tag{5.10}$$

Transmit

The energy consumed during transmission depends upon several factors: the length of time of the transmission, transmission frequency and the power used for broadcasting.

$$E_{trans,perbyte} = \Psi_{trans} t_{trans} \tag{5.11}$$

Here, Ψ_{trans} is device specific and should be measured empirically.

5.1.3 Exploring Different Transmission Trade-offs

Data can be transmitted at any stage along the pipeline, but there are distinct trade-offs in terms of energy and accuracy at each stage.

There are four ways of considering a system that senses and transmit data, distinguished by the point in the processing pipeline they transmit their data. The system can (1) transmit the raw data, (2) transmit calculated feature data, (3) transmit activity classifications, or (4) transmit a subset of activity classifications, defined as the subset of activities that are different than the previously transmitted activity. In other words, (4) only transmits activities when there is a transition from one activity to another.

Raw Data

Transmitting the raw data has highest accuracy and energy consumption.

$$E_{system} = E_{sense} + (E_{write,perbyte} + E_{read,perbyte} + E_{trans,perbyte}) \cdot D_{raw} \tag{5.12}$$

$$E_{system} = \Psi_{sense} t + (\Psi_{write} t_{write} + \Psi_{read} \cdot \frac{CO_{word}}{2F_O} + \Psi_{trans} t_{trans}) \cdot 2N \tag{5.13}$$

The energy for transmitting raw data grows linearly with time.

Features

Transmitting features can conserve energy (in comparison with transmitting raw features) if the amount of feature data is less than the raw data. Transforming the data from one space to another introduces some error.

For features, we introduce E_{feat} and D_{feat}. Naturally, these two variables vary greatly from feature to feature as seen above.

$$E_{system} = E_{sense} + E_{feat} + (E_{write,perbyte} + E_{read,perbyte} + E_{trans,perbyte}) \cdot D_{feat} \quad (5.14)$$

Now, let's look at the case when the features are scalars such as mean and variance. D_{feat} grows with the number of features, but we still divide by S_G. In this case E_{system} looks like:

$$\begin{aligned} E_{system} = & E_{sense} + \Psi_{cpu} \cdot N \cdot \frac{CO}{F_O} + (E_{write,perbyte} + \cdots \\ & E_{read,perbyte} + E_{trans,perbyte}) \cdot \frac{4N}{S_G} \end{aligned} \quad (5.15)$$

In this case, E_{system} can actually be reduced by increasing S_G. If we let S_G be a variable and set everything else then E_{system} is of the form:

$$E_{system} = \frac{\alpha}{S_G} + \beta \quad (5.16)$$

Now let's consider the case when the feature size depends on the grouping size, such as FFT.

$$\begin{aligned} E_{system} = & E_{sense} + \Psi_{cpu} \cdot N \log S_G \cdot \frac{CO}{F_O} + \cdots \\ & (E_{write,perbyte} + E_{read,perbyte} + E_{trans,perbyte}) \cdot 4N \end{aligned} \quad (5.17)$$

Letting S_G be a variable and all others constant, E_{system} now takes on the form:

$$E_{system} = \alpha \log S_G + \beta \quad (5.18)$$

The point here is that the choice of feature has a dramatic effect on the energy consumption of the system and how the system responds to changes in variables such as the grouping size.

Activity Classification

Transmitting only activity classifications can conserve energy when the energy required to calculate the classification and send the compressed result is less than the energy required to transmit features or raw data. The error introduced at this stage depends on the accuracy of the classification model used.

The advantage of transmitting only activity classifications is that you can drastically reduce the amount of data you need to send. You're effectively increasing the compression ratio. Here we introduce $E_{classify}$, which depends on the specific classification algorithm in use. In general, these are computationally complex operations, but the payoff you get is that you can compress data even further and thus reduce the data to transmit. We also introduce D_{act}. If we assume the activity classification can be represented in 2 bytes then $D_{act} = 2N/S_G$.

$$\begin{aligned} E_{system} =& E_{sense} + E_{feat} + E_{classify} + \cdots \\ & (E_{write,perbyte} + E_{read,perbyte} + E_{trans,perbyte}) \cdot D_{act} \end{aligned} \tag{5.19}$$

Now, for an example of a classifier, let's look at a support vector machine (SVM) with a radial basis function kernel, which is $O(S_G N_{SV})$ to evaluate, where N_{SV} is the number of support vectors.

$$\begin{aligned} E_{system} =& E_{sense} + E_{feat} + \psi_{cpu} S_G N_{SV} \frac{N}{S_G} \frac{CO}{F_O} + \\ & (E_{write,perbyte} + E_{read,perbyte} + E_{trans,perbyte}) \cdot \frac{2N}{S_G} \\ =& E_{sense} + E_{feat} + \psi_{cpu} N N_{SV} \frac{CO}{F_O} + \\ & (E_{write,perbyte} + E_{read,perbyte} + E_{trans,perbyte}) \cdot \frac{2N}{S_G} \end{aligned} \tag{5.20}$$

This case has good compression, but the energy used in calculation can be significant if the number of support vectors is high or if the classifier is more complex than a SVM. We think we can do a little better.

Transitions

Transmitting activity classifications only when transitions occur is another way of reducing the amount of data that needs to be processed and transmitted. It's based on the idea that a simple routine could act as a gatekeeper to the computationally complex classification algorithms and wireless communication if it can quickly determine if a change in activity has occurred. Two such transition detection strategies we've explored are a log-likelihood ratio test and using mean and variance.

We think we can reduce energy consumption by adding one more preprocessing step to determine if a transition has occurred between two sampling times. By looking for transitions, we only call the classifier if a transition has occurred. Now the number of times the classifier is called and the amount of data generated is dependent on the probability of transitions occurring, which we represent with τ, with $0 < \tau < 1$. This assumes we have a perfect transition detector. In reality τ depends on the true positives and false positives of the transition detector.

$$\begin{aligned} E_{system} =& E_{sense} + E_{feat} + E_{detect\_transition} + \tau E_{classify} + \cdots \\ & \tau(E_{write,perbyte} + E_{read,perbyte} + E_{trans,perbyte}) \cdot D_{act} \end{aligned} \tag{5.21}$$

Since the transition detector will be called more often than the classifier, one must ensure that $E_{detect\_transition} < E_{classify}$.

Our previous discussion of transition detection using a log-likelihood ratio test found a 5× decrease in power consumption with only a 5% sacrifice in accuracy [Boyd *et al.*, 2010]. A similar technique called Early Template Matching was employed by Raffa et. al. [Raffa *et al.*, 2010] in their gesture recognition system. The idea is to use a little processing to prevent even more processing from happening.

5.2 Strategies for Sensing Less

From the energy and data creation model we see that strategies for reducing energy consumption should focus on reducing the amount of data generated by sensing, thus reducing the amount of data transmitted. This section will focus on ways to smartly reduce sensing and the next section will cover reducing the amount of data to transmit.

5.2.1 Reduce F_S and bit resolution

Sometimes a smart way to reduce the data generated is to simply reduce the sampling frequency, F_S, of the sensors. For example, Krause *et al.* [2005] achieved nearly the same accuracy sampling accelerometers at 6Hz as they did at 20Hz, but improved the lifetime of their battery from 9.2 to 17 hours. The nature of the activity to be sensed should drive the choice of sampling frequency and the sampling frequency should be evaluated to see if it can be lowered while still meeting design specifications.

Another technique that doesn't exactly reduce the amount of sensing, but still has the affect of reducing the rate at which data is generated is the act of reducing the resolution, or number of bits, used to represent samples. Patel *et al.* [2009] saw a reduction in power consumption from $29.8\mu W$ to $12.5\mu W$ by reducing the bitwidth by 6 and observed only a 3% decrease in accuracy of their system. A higher bitwidth can measure smaller variations in a signal. If we are only interested in coarse changes in a signal, then a low bitwidth may be sufficient.

5.2.2 Periodically put sensors to sleep

Sensors are often sensing when nothing interesting is going on. Periodically putting the sensors to sleep will take advantage of those times, creating regular duty cycles of sleep and sense. However, this can introduce two types of error in the system:

latency and misses. For example, latency will be incurred when an important change in the signal occurs while the sensor sleeps, which the sensor does not detect until after it wakes up. Furthermore, important changes may be missed entirely while the sensor sleeps. Also, this scheme will still waste energy when it senses unnecessarily in scenarios where the changes in signal are sparse.

5.2.3 Aperiodically put sensors to sleep

To deal with the sparsity of some signals, techniques of aperiodically putting sensors to sleep have been developed. Aperiodically means sleep times will not be uniform in length; some intervals will be longer than others. These techniques seek to exploit the aperiodic nature of human activity and leverage the fact that humans move slowly compared to the speed at which modern microcontrollers operate. Examples of this are episodic sampling from Au *et al.* [2009], and sleep times based on a Gaussian distribution of activity length [French *et al.*, 2007; Krause *et al.*, 2005].

5.2.4 Use a hierarchy of sensors

Another way to reduce sensing is to selectively choose which sensors are powered. As some sensors consume more energy than others, we can use a hierarchy of sensors such that low-power sensors do most of the sensing and high-power sensors are turned on when necessary. This scheme has seen success with smartphones, which feature a heterogeneous mixture of sensors from low-power accelerometers and high-power GPS radios [Wang *et al.*, 2009; Paek *et al.*, 2010]. Even if sensors are homogeneous, in terms of the energy they consume, a hierarchy can be constructed which selectively turns off sensors that are not needed [Zappi *et al.*, 2008].

5.3 System Evaluation

Based on the analysis of our energy models, we were keen to see the models with values from a real wearable device, the TI Chronos Watch. It features a MSP430 digital signal processor, which includes a microcontroller, flash storage and a sub-1GHz wireless transceiver. Its sensors include a 3-axis accelerometer, altimeter, barometer, and thermometer. The device also has a LCD screen, buttons for interaction, and a small CR2032 coin-size battery. The device comes with a wristwatch-like enclosure so it can easily be worn on the wrist.

Several things make this an attractive test bed for the kind of software optimizations techniques we've talked about in this thesis. The small, unobtrusive form factor is as comfortable as a wristwatch. The small battery, limited on-board storage, wireless transceiver, and ability to turn off individual sensors make it suitable for testing how software can change how much energy the system uses.

We see from the equations in Section 5.1 that the grouping size has a big effect on energy consumed and data. We decided to investigate how the grouping size affects accuracy. We also looked into how much energy is consumed when the system transmits raw data, features, and activity classifications.

5.3.1 Accuracy

We begin with a simple simulation of how the grouping size, S_G, affects accuracy in terms of Precision, Recall, and F-Score. These three metrics were defined in Section 3.3. We've seen in this chapter that increasing S_G has energy saving benefits, but is there a limit to how high you can make the grouping size before it negatively affects the usefulness of the system? For this simulation we used the same data we used in the previous chapter, which is the synthetic data used to simulate lots of transitions.

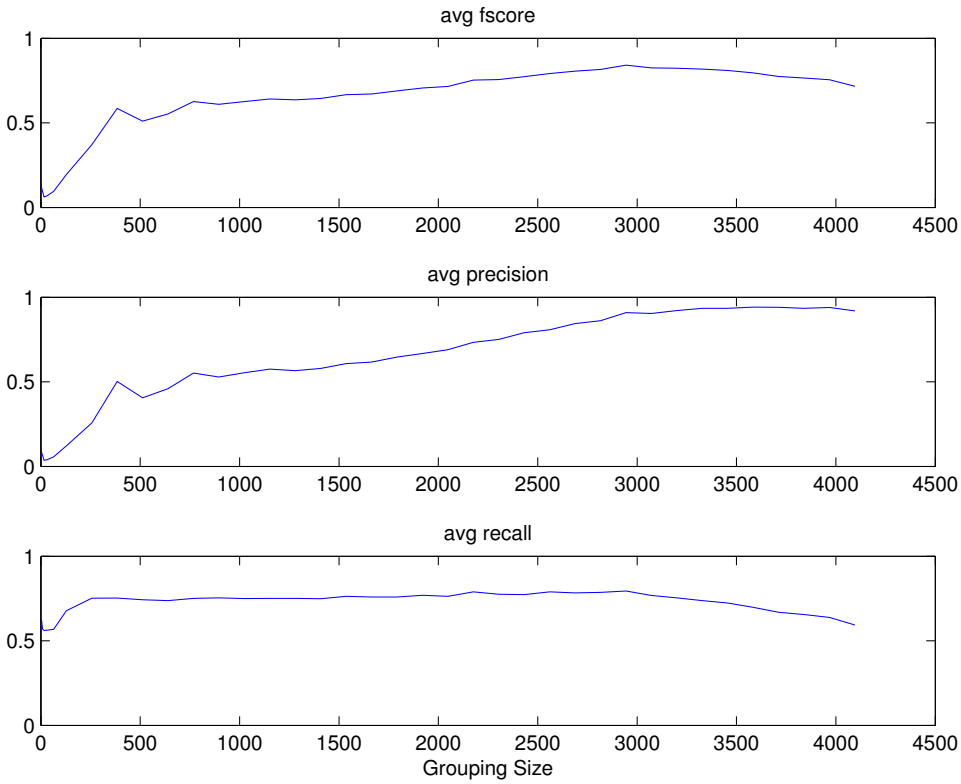

Figure 5.2: F-Score improves as the grouping size increase until the grouping size is approximately the mean value of the duration of activities. This simulation used the same data as the previous chapter, which is synthetic data used to represent many transitions. We assumed 100Hz sampling frequency and mean and variance transition detection. The results shown are the average of three different threshold values for mean and variance.

One important aspect about this data is that the average length of time for each activity is just under 30 seconds. We assumed a 100Hz sampling frequency and μ and σ^2 transition detection. This transition detection methods requires a threshold for μ and σ^2. We used the average of three different threshold values. The results can be seen in Figure 5.2.

F-Score increases until it peaks at a grouping size of approximately 3000 samples. Given that this data is sampled at 100Hz, that's about 30 seconds of time that each grouping represents. It is interesting to note that this corresponds with the average

Table 5.2: These are measurements of how much current the TI Chonos watch draws in different operating modes. The active mode uses the default 12MHz setting for the DSP and assumes no other sensor or radio is on. The accelerometer mode assumes a sampling frequency of 100Hz. The Active, Sleep and Read/Write Flash values were measured with a multimeter. The Accelerometer and Transmit values come from the TI datasheet. The TI Chronos watch runs on a 3V CR2032 coin-size battery.

Mode	Current (mA)
Sleep	0.0064
Active	5.4
Read Flash	2.9
Write Flash	2.2
Accelerometers	0.166
Transmit	35

activity length of the simulation data, which is also approximately 30 seconds. As S_G increases beyond the average activity length, the probability of miss, P_M, used in the calculation of Recall, increases greatly, thus bringing down F-Score.

5.3.2 The Energy Cost of Different Transmission Schemes

The choice of when to transmit data has a big impact on how much energy the system consumes. For this test we measured how much current the TI Chronos watch draws when operating in different modes. Table 5.2 summarizes these measurements. Notice that the amount of current the device draws in Transmit mode is much greater than Active or Sleep.

We were most curious to know how much energy could be saved over the course of a day given four scenarios:

1. Transmit the raw data

2. Transmit features and compare a scalar feature versus a vector feature

3. Transmit after activity classification; include energy required for classification

4. Transmit activity only after transition detection

We set up a simulation in Matlab that assumed the TI Chronos watch would be active for 12 hours and sensing the accelerometers at 100Hz. We further assumed we had a 100% accurate transition detector and that the probability of a transition occurring was 0.50. A probability of a transition occurring of 0.50 would mean that about half the time the transition detector runs it would detect a transition and call the classifier. A value of 0.50 is probably high, but it should account for false positives that would occur in the real world. We assumed the classifier to be a SVM which has computational complexity based on the number of support vectors and input values. A grouping size of 128 values was chosen. To see the difference in the choice of feature, we ran two simulations to show the difference between calculating mean and variance (two scalar features) and the FFT (a vector feature). The results can be seen in Figure 5.3. Note the two bar graphs are on the same horizontal scale. The amount of energy to sense the accelerometers is the same for both situations and is omitted from this figure.

The breakdown of how much energy classification and transition detection use is not that important compared to the savings evident in reducing the amount of data generated and transmitted. The compression ratio that activity classification achieves significantly reduces the amount of energy the entire system would use over the course of day. Transition detection enables only a slight gain over just classification. Transmission detection would become more important if the energy to classify were significantly more than the energy to detect a transition. This simulation assumed we used a SVM to classify, which is relatively inexpensive, computationally, compared to other classifiers used for activity recognition such as Hidden Markov

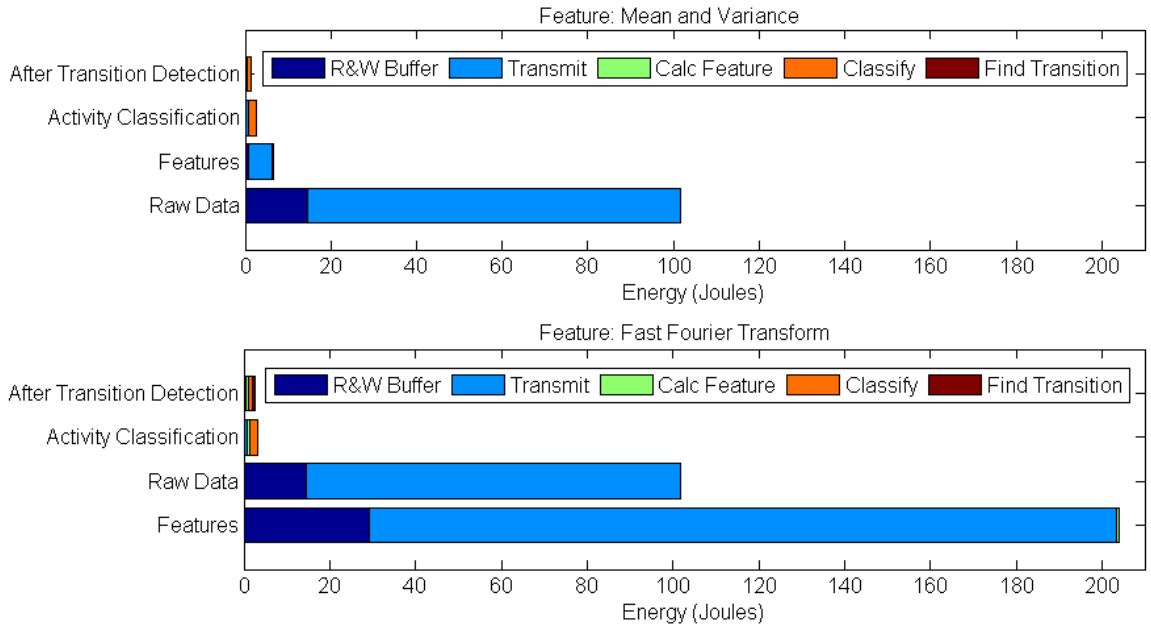

Figure 5.3: This figure shows the energy savings of data compression and the choice of when to transmit data. These two graphs show the energy used for a simulated situation on the TI Chronos watch that continuously senses the accelerometers at 100Hz and calculates either mean and variance in the top graph or FFT in the bottom graph. Current and voltage measurements for this simulation were taken from the measurements in Table 5.2. The important thing to note here is the significant energy savings that activity classification enables by reducing the amount of data that needs to be stored and transmitted. Feature calculation has little energy overhead compared to the energy required to store and transmit data.

Models, Bayes Nets, or K-Nearest Neighbor. The choice of feature is also very evident in this figure. It is interesting to see that the energy overhead of calculating features pales in comparison to the energy required to store and transmit data. These measurements are more evidence of how energy efficient CPUs are in comparison to file I/O.

5.4 Summary

Our models of energy and data usage show how just a few design parameters in a wearable activity recognition system can affect its energy consumption and accuracy. The models showed the importance of the grouping size and choice of feature. For

scalar features, the energy consumption of a system decreases as the grouping size increases. For vector-based features, the energy increase with the log of grouping size. The models also show that activity classification can significantly compress the data and our experiments verified that classification greatly reduced the energy a system would consume in a day. We have also shown that for the TI Chronos watch, the energy overhead of feature calculation and activity classification is minuscule compared to the saving gained by not storing or transmitting that data.

Chapter 6

CONCLUSION

6.1 Contributions

This thesis has contributed the following to the study of how software affects the energy vs. accuracy trade-off in a wearable activity recognition system:

1. The idea of using transition detection as a means of reducing the number of times activity classification algorithms are called. We explored three different techniques of transition detection: the log-likelihood ratio test, support vector machine, and mean-and-variance thresholds. We found the design space surrounding transition detection to be quite large when you take into account the choice of sensors, features, and temporal parameters. In general we found that simple scalar features work well for the task and are more energy efficient than vector-based features.

2. An exploration of the energy consequences of design choices in each of the major stages in activity recognition: sensing, sleep schedule, feature extraction, classification, and storage or transmission.

3. A framework to answer the question of when it is advantageous to include transition detection in a system, given the probabilities of transitions occurring and the energy characteristics of a platform.

4. The introduction of Hybrid Episodic Sampling, a sleep schedule that incorporates activity transition detection with a sleep schedule that changes based on

the activity classifications. This schedule was shown to use less energy and be more accurate than a uniform sleep strategy.

5. An explanation of the "Great Compromise": a situation in which some small sacrifice in overall accuracy results in substantial gains in energy efficiency. The fact that this phenomenon occurs should motivate systems designers to carefully consider the parameters of their system and search for ways to optimize for energy without sacrificing too much accuracy.

6. A mathematical model of energy consumed and data generated in each stage of the activity recognition process. This analysis showed the importance of the grouping size, which is the number of samples your system treats as a block to analyze. It also demonstrated the importance of reducing the amount of data generated, stored and transmitted. Sensing and wireless transmission consume the most amount of energy per byte, so optimizations that reduce the amount of time sensing and reduce the amount of data to be transmitted greatly reduce the energy a system consumes.

These ideas were tested using two embedded systems: a SpakFun IMU and a TI Chronos watch, though much of the analysis of different situations was done using Matlab. The work provided in this document shows compelling evidence that software has a profound effect on the compromise of energy consumption and accuracy. This work also shows that incorporating knowledge of human behavior into the design of the system can help it be more energy efficient without sacrificing too much accuracy.

6.2 Future Work

A full implementation of these ideas running on an embedded system and a complete trial with human subjects remains to be done. It remains to be seen if transition

detection via mean and variance, SVM, or a log-likelihood ratio test will generalize and scale to multiple users. It would also be instructive to see multiple machine learning algorithms implemented in an embedded system like the TI Chronos Watch, so that the energy cost of each algorithm could be compared against the energy cost of transition detection.

There is also the question of how Jevons Paradox can be avoided. Jevons Paradox is the phenomenon that an increase in energy efficiency creates more demand for fuel rather than reducing demand. In a general sense, Jevons Paradox can be avoided if a technology, for example, doubles in efficiency and, at the same time, does not double the demand for the energy source. The social and political ramifications of Jevons Paradox for computing platforms could be explored.

There is also the growing field of energy harvesting to consider. An embedded system that employs energy harvesting would supplement or rely exclusively on energy from the surrounding environment. Some examples include solar power, kinetic energy transformed to electric energy via piezoelectric resistors, and electromagnetic energy from radio and television broadcasts. Imagine a finely-tuned system that could operate in any number of energy profiles based on the current and projected energy consumption rates and energy storage. Energy profiles would consist of different permutations of the variables in the design space: sampling frequency, sensors, features, grouping size, classifier, etc. The controlling software could switch between energy profiles as energy availability fluctuates much as online video streaming services can switch between different quality of audio and video based on current bandwidth availability.

REFERENCES

Au, L., M. A. Batalin, T. Stathopoulus, A. A. T. Bui and W. J. Kaiser, "Episodic Sampling: Towards Energy-efficient Patient Monitoring with Wearable Sensors", in "31st Annual International Conference of the IEEE EMBS", (Minneapolis, MN, USA, 2009).

Avci, A., S. Bosch, M. Marin-Perianu, R. Marin-Perianu and P. Havinga, "Activity Recognition Using Inertial Sensing for Healthcare, Wellbeing and Sports Applications: A Survey", in "23rd International Conference on Architecture of Computing Systems, ARCS 2010", (Hannover, Germany, 2010).

Bao, L. and S. Intille, "Activity Recognition from User-Annotated Acceleration Data", in "Pervasive 2004", pp. 1–17 (Vienna, Austria, 2004).

Bharatula, N. B., M. Stäger, P. Lukowicz and G. Tröster, "Empirical Study of Design Choices in Multi-Sensor Context Recognition Systems", in "IFAWC: 2nd International Forum on Applied Wearable Computing", (Zurich, Switzerland, 2005).

Boyd, J., H. Sundaram and A. Shrivastava, "Power-Accuracy Tradeoffs in Human Activity Transition Detection", in "Design, Automation, and Test in Europe. DATE 2010.", (Dresden, Germany, 2010).

Chang, C. C. and C. J. Lin, "LIBSVM: a library for support vector machines. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm", (2001).

Cristianini, N. and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods* (Cambridge University Press, 2010).

Czabke, A., S. Marsch and T. C. Lueth, "Accelerometer based real-time activity analysis on a microcontroller", in "Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2011 5th International Conference on", pp. 40–46 (2011).

French, B., D. Siewiorek, A. Smailagic and M. Deisher, "Selective Sampling Strategies to Conserve Power in Context Aware Devices", in "Proceedings of the 2007 11th IEEE International Symposium on Wearable Computers", (Boston, MA, USA, 2007).

Furber, S. and D. Brown, "INTERVIEW A Conversation with Steve Furber", Queue **8**, 2, 1–8, URL `http://doi.acm.org/10.1145/1716383.1716385` (2010).

Gebruers, N., C. Vanroy, S. Truijen, S. Engelborghs and P. P. De Deyn, "Monitoring of physical activity after stroke: a systematic review of accelerometry-based measures", Archives of physical medicine and rehabilitation , 2, 288–297 (2010).

Havinga, P. and G. J. M. Smit, "Design Techniques For Low Power Systems ", Journal of Systems Architecture **46** (2000).

Huynh, T., M. Fritz and B. Schiele, "Discovery of Activity Patterns using Topic Models", (2008).

Krassnig, G., D. Tantinger, C. Hofmann, T. Wittenberg and M. Struck, "User-friendly system for recognition of activities with an accelerometer", in "4th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth)", pp. 1–8 (2010).

Krause, A., M. Ihmig, E. Rankin, D. Leong, S. Gupta, D. Siewiorek, A. Smailagic, M. Deisher and U. Sengupta, "Trading off Prediction Accuracy and Power Consumption for Context-Aware Wearable Computing", in "9th IEEE International Symposium on Wearable Computers ISWC'05", (Osaka, Japan, 2005).

Lorch, J. R. and A. J. Smith, "Software Strategies for Portable Computer Energy Management", IEEE Personal Communications (1998).

Metcalfe, R. M. and D. R. Boggs, "Ethernet: distributed packet switching for local computer networks", Communications of the ACM **19**, 395–404 (1976).

Mitchell, T. M., *Machine Learning* (McGraw-Hill, New York, 1997).

Nyan, M., F. Tay, K. Seah and Y. Sitoh, "Classification of gait patterns in the time-frequency domain", Journal of biomechanics **39**, 14, 2647–2656 (2006).

Paek, J., J. Kim and R. Govindan, "Energy-Efficient Rate-Adaptive GPS-based Positioning for Smartphones", (2010).

Patel, K., C. Chern-Pin, S. Fau and C. J. Bleakley, "Low power real-time seizure detection for ambulatory EEG", in "Pervasive Computing Technologies for Healthcare, 2009. PervasiveHealth 2009. 3rd International Conference on", pp. 1–7 (2009).

Raffa, G., L. Jinwon, L. Nachman and S. Junehwa, "Don't slow me down: Bringing energy efficiency to continuous gesture recognition", in "Wearable Computers (ISWC), 2010 International Symposium on", pp. 1–8 (2010).

Rivera, J. and R. van der Meulen, "Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020", URL `http://www.gartner.com/newsroom/id/2636073` (2013).

Roy, K. and M. C. Johnson, "Software Design for Low Power", in "Low power design in deep submicron electronics", (Kluwer Academic Publishers, 1997).

Sekine, M., T. Tamura, T. Togawa and Y. Fukui, "Classification of waist-acceleration signals in a continuous walking record", Medical Engineering & Physics **22**, 4, 285–291 (2000).

Smit, G. J. M. and P. Havinga, "A survey of energy saving techniques for mobile computers", Tech. rep., Twente, The Netherlands (1997).

Stäger, M., P. Lukowicz and G. Tröster, "Power and accuracy trade-offs in sound-based context recognition systems", Pervasive and Mobile Computing **3**, 3, 300–327 (2007).

Stikic, M., T. Huyng, K. V. Laerhoven and B. Schiele, "ADL Recognition Based on the Combination of RFID and Accelerometer Sensing", (2008).

Sun, F.-T., C. Kuo and M. Griss, "PEAR: Power efficiency through activity recognition (for ECG-based sensing)", in "Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2011 5th International Conference on", pp. 115–122 (2011).

Tiwari, V., S. Malik and A. Wolfe, "Instruction Level Power Analysis and Optimization of Software", Journal of VLSI Signal Processing **13** (1996).

Uswatte, G., C. Giuliani, C. Winstein, A. Zeringue, L. Hobbs and S. L. Wolf, "Validity of accelerometry for monitoring real-world arm activity in patients with subacute stroke: evidence from the extremity constraint-induced therapy evaluation trial", Archives of physical medicine and rehabilitation **87**, 10, 1340–1345 (2006).

Wang, Y., J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari and N. Sadeh, "A Framework of Energy Efficient Mobile Sensing for Automatic User State Recognition", (2009).

Zappi, P., C. Lombriser, T. Stiefmeier, E. Farella, D. Roggen, L. Benini and G. Tröster, "Activity Recognition from On-Body Sensors: Accuracy-Power Trade-Off by Dynamic Sensor Selection Wireless Sensor Networks", in "Wireless Sensor Networks", edited by R. Verdone, vol. 4913, pp. 17–33 (Springer Berlin / Heidelberg, 2008).

Zotos, K., A. Litke, A. Chatzigeorgiou, S. Nikolaidis and G. Stephanides, "Energy Complexity of Software in Embedded Systems", (2005).

APPENDIX A

THE COMPUTATIONAL COMPLEXITY OF LLRT

This appendix describes the derivation of the ratio test's complexity. Let D be the dimensionality of the feature we are analyzing. Let N be the number of frames, or observations, per window. Calculation of the probability p in Equation 3.1 involves calculating the mean feature vector (μ) and covariance matrix (Λ) across the entire window. The complexity of μ is $O(DN)$, because features are calculated per frame and each dimension of the feature is averaged. The complexity of Λ is $O(D^2N)$. The calculation of p also involves calculating Λ^{-1}, which has complexity $O(D^3)$ since the best known algorithms to calculate the inverse of a matrix are cubic. The complexity for calculating p can now be simplified to $O(D^3) + O(D^2N)$.

APPENDIX B

TRANSITION DETECTION TESTS WITH PARETO FRONT

The following four figures show all the results for each of the three transition detection methods, and no transition detection at all. The color and shape represent different sampling frequencies and each point represents a permutation of each strategy's variables. The Pareto-front is highlighted. These points represent the optimal points for which there no more accurate combination given some level of energy consumption.

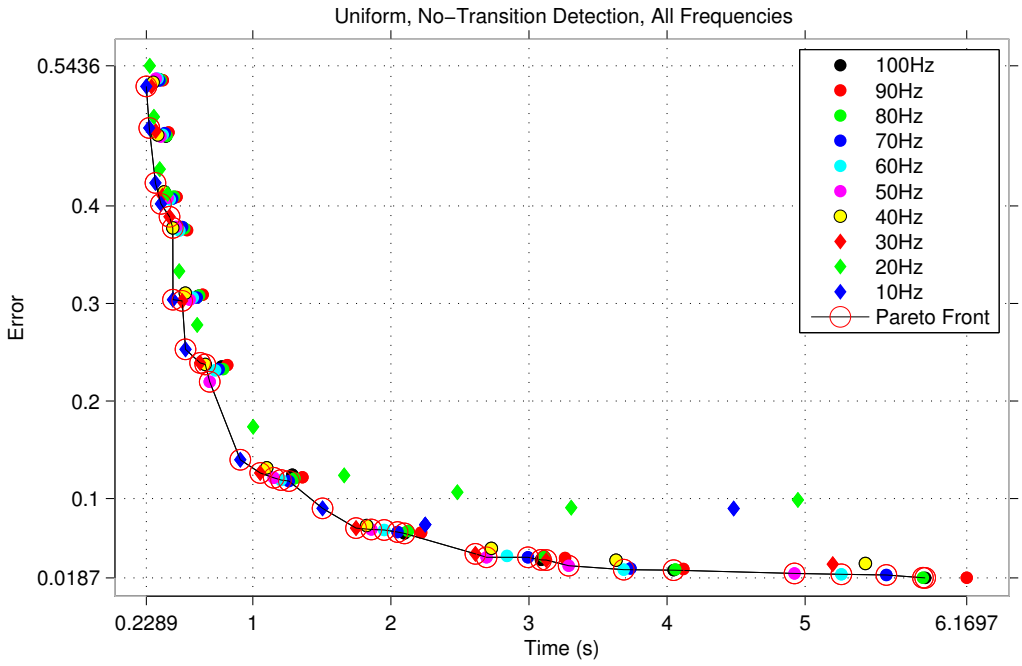

Figure B.1: For the uniform, no-transition detection strategy, the only variable is t_{max}. The cluster of points near the "knee" of the curve all have t_{max} values of 8s. It's interesting to note in this figure that sampling frequency doesn't have a significant impact on error until 20Hz.

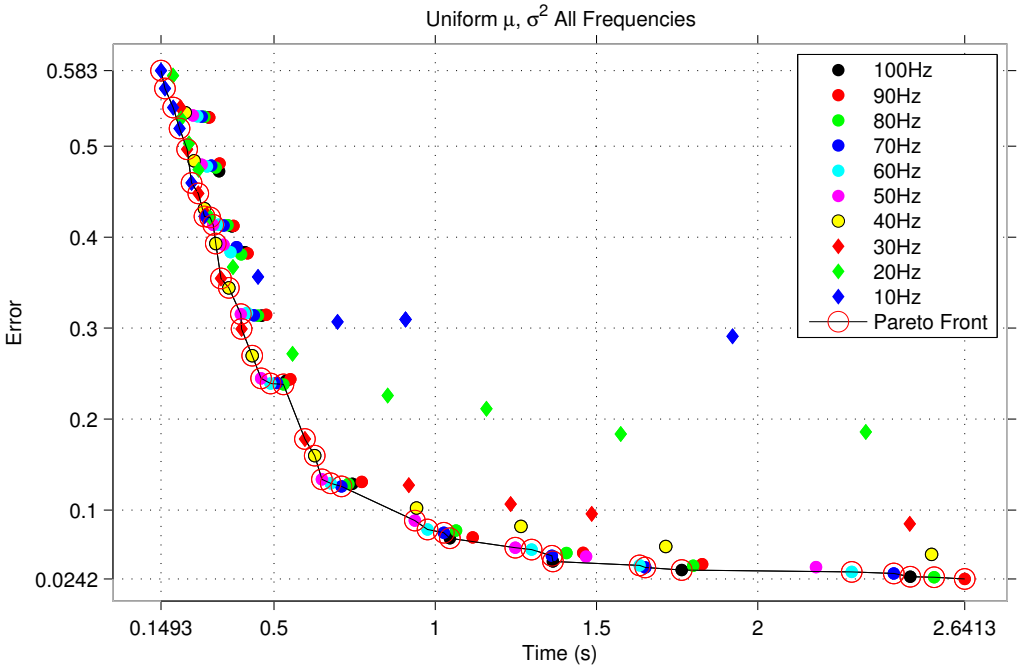

Figure B.2: For transition detection with μ and σ^2, the variables are t_{max} and the threshold values for μ and σ^2. This figure shows the result of performing a grid search on the two threshold values and finding the threshold values that offer the best trade-off between power and accuracy. It's also interesting to note that the impact of sampling frequency on accuracy is evident at a higher sampling frequency than no transition detection. The cluster of points near the "knee" of the curve have t_{max} values of 8s.

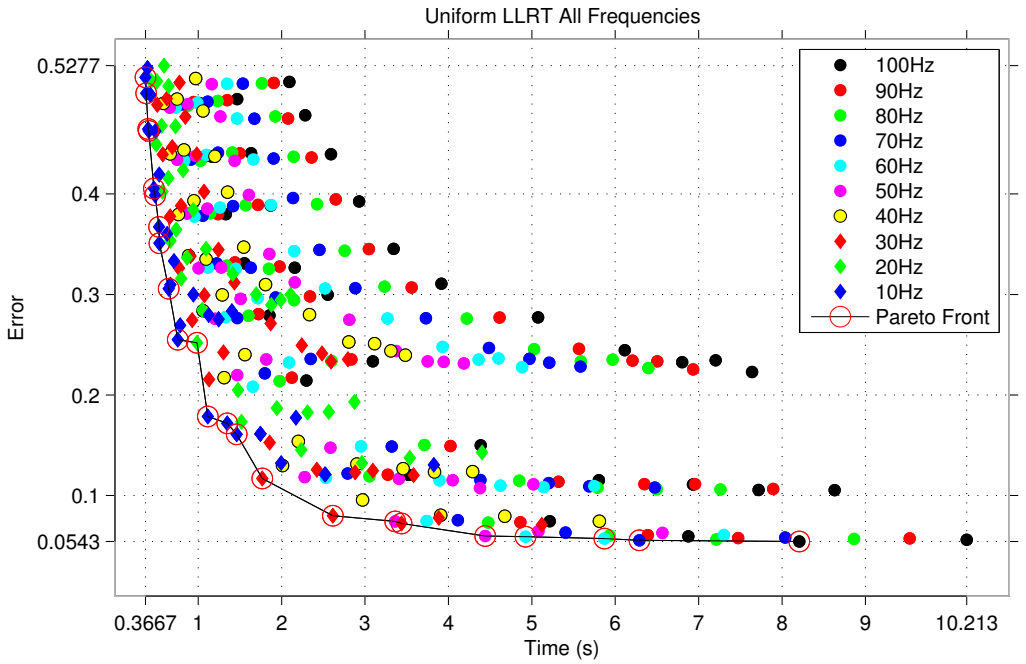

Figure B.3: For transition detection with LLRT, the variables are t_{max}, the length of the analysis window, and the LLRT threshold value. Here, the effect of sampling frequency is more pronounced and the lowest sampling frequencies lose little in accuracy compared to the higher frequencies.

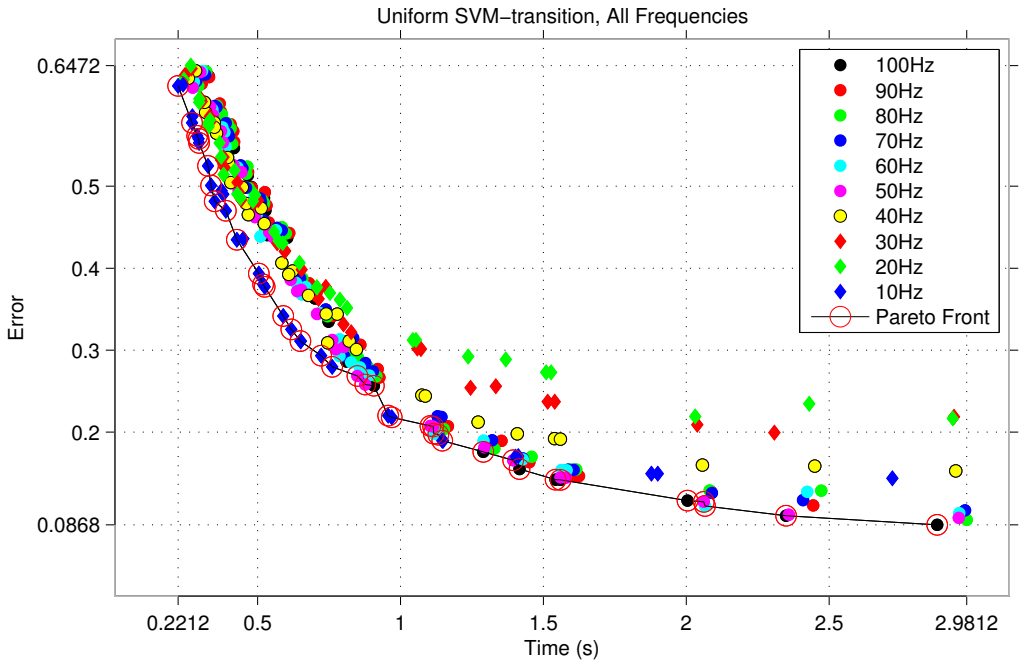

Figure B.4: For transition detection using a SVM, the variables are t_{max} and the size of the analysis window. It is interesting to note that the size of the analysis window has little effect on the energy consumption compared to LLRT. There is also a big jump in sampling frequency in the Pareto-front near the "knee"; it goes from 100Hz to 10Hz.