A Framework for the Automated Generation of Power-Efficient Classifiers for Embedded Sensor Nodes

Ari Y. Benbasat and Joseph A. Paradiso
The Media Laboratory
Massachusetts Institute of Technology
{ayb,joep}@media.mit.edu

Abstract

This paper presents a framework for power-efficient detection in embedded sensor systems. State detection is structured as a decision tree classifier that dynamically orders the activation and adjusts the sampling rate of the sensors (termed groggy wakeup), such that only the data necessary to determine the system state is collected at any given time. This classifier can be tuned to trade-off accuracy and power in a structured, parameterized fashion. An embedded instantiation of these classifiers, including real-time sensor control, is described.

An application based on a wearable gait monitor provides quantitative support for this framework. The decision tree classifiers achieved roughly identical detection accuracies to those obtained using support vector machines while drawing three times less power. Both simulation and real-time operation of the classifiers demonstrate that our multi-tiered classifier determines states as accurately as a single-trigger (binary) wakeup system while drawing as little as half as much power and with only a negligible increase in latency.

Categories and Subject Descriptors

I.5.2 [**Pattern Recognition**]: Design Methodology; C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems

General Terms

Algorithms, Design, Performance

Keywords

tiered wakeup, power-efficient detection, dynamic power management, wearable sensors

1 Introduction

Embedded sensor nodes are currently being used in a wide array of applications. These include, but are certainly not limited to, detecting degenerative diseases [3], monitoring remote regions [29], and ensuring the safety of housebound elders [15]. Such systems are part of a new class of sensor-driven applications, leveraging the decrease in both price and size of components to allow rich, multimodal data streams to be captured by very compact systems.

However, as sensors nodes increase in functionality, they require more frequent activation and therefore more frequent replacement or recharging of batteries. This creates an increasing gap between the capabilities of a device and its lifespan under normal use. Thus, current applications of embedded sensor systems are mostly limited to prototype and experimental usage or very simple implementations. The most common solutions to limited lifespan are to tether the system to wall power [21] or to confine the system to sampling at such a low rate that the lifespan is satisfactory [34]. Obviously, the full potential of wireless and wearable sensors is not being achieved through such systems, as their limited lifespans, sensing capabilities or update rates greatly reduce the utility to the end-user.

By concentrating our design efforts on the sensors themselves, rather than on the networks, it is possible to construct a class of embedded systems which achieve their sensing goal(s) while drawing significantly less power. This will increase the lifespan of embedded sensor nodes, allowing many more applications to make the transition from laboratory to marketplace and thereby benefit a much wider population.

1.1 Approach

This work improves the capability/lifespan gap in embedded sensor nodes through high-level algorithmic means rather than low-level technical ones. We started from a fundamental: the *raison d'etre* of these devices is to collect and process data and therefore the design of the sensors themselves should be central. We concentrated on reducing the energy usage of the sensors within the nodes. This metric was chosen since it is both general and tractable, though it is important to note that any power savings in the form of reduced sensing also correspond to further power savings through a reduction in the data to process, the data to transmit or store and the data to analyse. Further, any gains through this work can be considered independently from the large body of work exploring power savings through improvements to the software [31], hardware [1], and RF efficiency [23] in wireless sensor networks.

Our goal is to determine the system state at any given point in time for the smallest outlay of energy. Specifically,

SenSys'07, November 6–9, 2007, Sydney, Australia.

System	Baseline		Sensing		
	Processor	RF Monitoring	Type	Power	% Power for Sensing
Gait Shoe [3]	35 mW	N/A	IMU every 5 ms	65 mW	65
ZebraNet [38]	15 mW	13 mW	GPS every 8 min	30 mW	52
Great Duck Island [32]	118 μW	465 μW	Ambient every 5 min	118 μW	17

Table 1. Power usage breakdown of selected embedded nodes/networks

the power drawn by the sensor node is reduced by dynamically adjusting the activation and sampling rate of the sensors, such that only the data necessary to decide the system state is collected at any moment. Overall, the amount of data gathered by the system is reduced without affecting the amount of useful information collected.

The form of our solution is such that the sensor sampling rates, as well as the transitions between them, are generated in a semi-autonomous fashion and can easily be embedded in hardware. The solution is also parameterized to allow for tuneable power/accuracy trade-offs. Therefore, this work should be applicable to a wide variety of applications.

1.2 Relevance

While the processor and RF transceiver are by far the largest power drains in long-range wireless sensor networks, sensor power usage is often on par with them in light-weight and wearable instantiations [28]. The power usage of three field-tested wireless sensor systems is examined.

The Gait Shoe is a wearable medical sensor for collecting information about a patient's manner of walking. It is centred on an inertial measurement unit (IMU) sampled at 200 Hz, with the collected data streamed wirelessly to a basestation. ZebraNet is a wireless sensor network composed of collared zebras. The core components of each collar are a long-range radio and a GPS unit sampled once every eight minutes. The zebras themselves comprise a mobile peer-to-peer network whose goal is to aggregate the sensor readings from all the units at each node. Finally, the Great Duck Island habitat monitoring project (GDI) was a wireless sensor network designed to track the Storm Petrels which populate the eponymous island. Sensor nodes were placed at the entrance of the birds' nests to record their comings and goings through measurement of the humidity and ambient temperature every five minutes. This data was sent to a gateway node which was the first level of a hierarchical network which eventually transmitted the data off the island.

For each project, power usage is broken down into two categories – the baseline power to run the processor and the wireless link and the power expended in sensing – and is summarized in Table 1. The baseline power varies with the level of networking in each application. Since the Gait Shoe is part of a hub and spoke network, it does not monitor an RF channel and therefore has low baseline power usage. ZebraNet uses a moderate amount of energy for networking — although the radio consumes high power, it is rarely used. By contrast, the GDI radio is relatively low power but is very frequently powered up to listen for messages, and therefore uses almost four-fifths of the static power draw. As for the power usage of the sensing, the gait shoe samples a half dozen sensors at a high rate and ZebraNet samples a single high power sensor at a low rate, drawing approximately half the total power in each case. By contrast, the Great Duck Island project collects small amounts of data at a low rate, and therefore dedicates only one sixth of its power to sensing.

Overall, each system devotes a significant percentage of its power usage to collecting sensor data. Further, the power used to monitor the radio channel in the two network applications is partially proportional to the amount of sensor data transmitted over the network (nodes also listen for instructions and network maintenance). Therefore, a reduction in sensing should lead directly to a reduction in both RF transmitter and receiver power usage.

2 Solution Overview

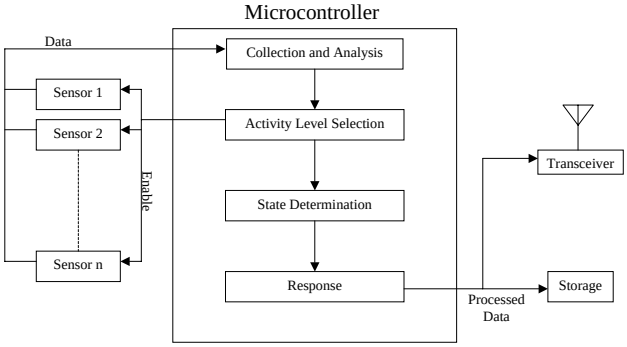

Figure 1. Flowchart of hardware solution

The general design for our system, shown in Figure 1, is centred around the concept of **tiered**, or "groggy", **wake up**. This stands in contrast to the more common binary wake up systems, which have only two modes: *fully active*, collecting all possible data and drawing maximal power, or *fully asleep*, collecting no data and drawing virtually no power. Instead, we envision a system with a number of different levels of activity and associated power usages. Each level comprises the currently active sensors for state determination as well as their sampling rate, together with algorithms to describe the level transitions. At each time step, the current activity level will specify how to use as little energy as possible to determine the current state and whether a level transition is necessary.

For a given application, the system states are designated by the designer. Each state represents an interesting condition of the system — *e.g.* a certain gait for a wearable medical device or the presence of certain fauna for an environmental node. Further, the application designer can associate specific responses, such as data capture and storage, with individual states.

The design process (and this document) proceed as follows. First, hardware for the desired application is config-

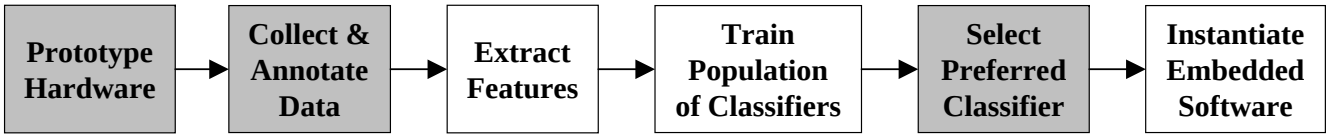

Figure 2. Framework workflow (automated steps unshaded)

ured for testing (section 4). A training data stream is collected with this hardware, and is annotated by the application designer. These annotated examples are used to construct a classifier that will determine the system state (section 5). The sensor set used by the classifier allows the final, possibly pared down, form of the hardware to be built and the state classifier to be implemented on it (section 6). Figure 2 shows the major steps in the framework. Those requiring intervention by the application designer are shaded.

3 Related Works

There are a number of related projects with the same overarching goal as our own — the reduction of power usage in embedded nodes by controlling sensor sampling. They can be divided based on their application: data collection, state determination, or tracking.

Most concentrate on the measurement and collection of data from a single phenomenon. Jain and Chang [17], Liu *et al.* [22] and Rahimi *et al.* [29] each use a different model of the sensor data to adjust the sampling rate of a sensor on a single node. Jain and Chang use the innovation of a Kalman filter [13] as a measure of the entropy rate of the data stream and adjust the sampling rate accordingly. Liu *et al.* model the system as a random walk and vary the sampling rate when subsequent measurements fall outside of the expected range. Finally, Rahimi *et al.* measure a phenomenon over a fixed area and use the spatial rate of change of the data to adjust the spacing of the samples. These are unsupervised approaches — more data is collected because the phenomena is varying at a faster rate. While these techniques generated good results in sample applications, they assume that data should be collected in all states and cannot differentiate between them. Further, in the case of Jain and Chang, the Kalman filter is a fairly structured and computationally expensive model that would not be appropriate for all systems.

He *et al.* [14] and Zhao *et al.* [39] consider power savings in tracking networks with the goal of reducing the total power usage of the nodes. He *et al.* solve the problem through the use of sentry nodes — a subset of the network that continuously monitors for events — which awaken the other nodes when there is an interesting phenomenon in the vicinity. Zhao *et al.* examine the problem in the context of a query from an individual node, which then queries other nodes along a gradient to acquire more accurate data. In selecting the path for the request to travel, the system takes into account both the expected utility of the information to be collected and the power necessary to collect the data and transmit the results back to the requesting node. While these techniques save significant power by leaving most of the network asleep at any given time, they respond to information in a strictly binary fashion. He *et al.* turn on all the neighbouring nodes on an event trigger, without consideration of the amount of additional data necessary to accurately track the phenomenon. Similarly, in Zhao *et al.*'s network, nodes selected as the next hop along the gradient do not use any knowledge of their circumstances to determine whether it is worthwhile for them to collect data or if the request should simply be passed along to a more suitable node.

Yu *et al.* [37] and Dutta *et al.* [11] consider power savings in the more general context of state detection. Yu *et al.* examine the case of a set of independent nodes in a network dedicated to making a binary state decision. Each node collects samples until a decision can be made with sufficient accuracy, with these decisions fused at a central node. Dutta *et al.* examine the case of a multi-sensor node tasked to determine the form (civilian, soldier or vehicle) of nearby objects. These nodes use a binary wakeup scheme, where a thresholded infrared sensor triggers the more expensive acoustic and magnetic sensors to collect enough information to make a decision. Both projects reduce power usage for state detection using the concept suggested in this framework — actively considering in real-time which information is necessary to make a decision — though in a more limited fashion. The sequence of measurements in Yu *et al.* is always taken with the same sensors set at the same rate, without considering the information necessary from *each* sensor or a model of the time evolution. The system in Dutta *et al.* wakes all sensors based on a single trigger, when a subset of the sensors is enough to make certain determinations. Incorporating more information about the system of interest could result in power savings in each case.

Other system properties bear discussion. First, as mentioned above, most of the projects either do not model the phenomena of interest or use a trivial model thereof, eschewing possible power savings. Second, most systems do not vary sensor usage beyond an all-or-nothing approach. Only Dutta *et al.* construct a hierarchy of sensors, and only the data collection applications consider varying the sensor sampling rate. Finally, there is a paucity of published details both regarding derivations of the devised algorithms and about how to generate an instantiation for a specific problem. Since the works referenced were published either in conference proceedings or magazines, it is difficult to determine whether this is an omission for space or if the construction was in fact *ad-hoc*.

A number of important works do not fall neatly into the categories above. Many projects consider power/accuracy trade-offs as part of the offline decision process. For instance, looking at wearable examples, Bao and Intille [4] demonstrate that for human activity recognition based on body-worn accelerometers, only sensors on the thigh and wrist are necessary, with other positions giving only marginal increase in accuracy (Lester *et al.* [20] confirms

this result). In terms of model-based systems, Deshpande *et al.* [9] present a very detailed solution where queries from a root node are designed to minimize power usage when executed in the network. The observation plan specifies not only the nodes to visit but the individual sensor(s) to be sampled at each, and the optimization is based on both the cost of the sensing and of data transmission. This work is excluded from the above discussion because the observations are centrally planned and do not vary based on measured data.

4 Hardware Platform

To simplify the rapid prototyping and testing of embedded sensor nodes, we designed a modular sensor platform. This platform is based around a series of circuit boards, each of which instantiates a specific sensing modality — *e.g.* inertial sensing, tactile sensing or ambient sensing — or portion of the sensing infrastructure — *e.g.* data collection, data processing or wireless communication. These boards can be arbitrarily combined and recombined, allowing for the design to quickly consider various proposed sensor combinations. Full details of this system can be found in [6]. Only two new design techniques used to improve the power efficiency of the boards for use with this framework are considered here.

The first technique is the use of parts with short wake up time. Since much of the power savings from the hardware design is predicated on power-cycling the various components, reducing the wake up time is key to minimizing the power wasted during that interval. This parameter can vary widely both between different sensing mechanisms for a given phenomenon (*e.g.* effectively nil for a phototransistor to 40 ms for a IR rangefinder) and individual parts (*e.g.* 8 ms for the ADXL202 MEMS accelerometer to 100 ms for the pin compatible MXR2312 thermal accelerometer). The wake up time sets the upper limit on how quickly a sensor can be cycled while still offering power savings over continuous activation. It should be noted that the availability of this information is spotty at best — provided on some data sheets while completely ignored on others. Further, no information is given about the power draw during wakeup. In many cases it is likely the same as during normal operation, though for some sensors (*e.g.* those that need to charge internal capacitors or equilibrate filters), it may well be significantly more.

A second key design technique is the use of multiple sensors to measure a single parameter of interest. The vast majority of sensor systems limit themselves — usually in the interests of simplicity or compactness — to a single sensor for each modality of interest. No matter how efficient such an implementation is for extracting information, it is guaranteed to be power inefficient in states where less (or more) data is necessary to determine the transitions. A system which can tailor its sensing in real-time to the current state of the device can draw far less power on average. While it seems counter-intuitive to make a system more power-efficient by adding complexities (and/or redundancy), the key is that the system has been given a new, lower energy source of information.

5 Selection and Design of Classifier

5.1 Data Collection

Prior to training the classifiers, a set of examples must be collected. Each example is a series of values with a given label (in this case, the state). The goal of the classifier is to create a mapping between the values and the states. For the classifier to do so as accurately as possible, it should be trained with examples spanning both the range and variation of the possible states. Further, those states need to be labelled as accurately as possible. Any sensors which could possibly be useful to the classification process should be included, as those not used by the classifier can always be removed in a later revision of the hardware. Data streams are captured at their maximum useful data rate.

To avoid data collection becoming an open-ended process, a scripted sequence is used to quickly acquire the most relevant data. The sequence is reasonably short and collects a suitable set of training data (dependant on the application) containing the active (high energy/variance) states, both interesting and uninteresting, that are known to the designer. Rather than wait for these states to occur naturally in the operation of the system, it is most often easier to simply manufacture them. This guarantees their presence, quality and labelling. If desired, long-term background recording can also be captured to provide a baseline for the uninteresting cases and to catch states which were not considered by the designer. This stream would therefore provide a measure of completeness which is lacking from the scripted stream. While it is possible to use such a stream as the sole source of the training data, this tends to be inefficient since the length of the recording necessary to acquire sufficient good examples of all complex states would be quite long and the vast majority of the remaining data will be of little to no value in the classifier training. Studies by Intille *et al.* [16] support this type of data collection scheme.

We have chosen this offline supervised training approach based on the assumption of fairly constrained applications and clarity of designer intent. The application designer will hand annotate the data streams, labelling the areas containing different types of interesting data. The remaining data is assumed to not be of interest and is grouped together. This allows the designer to explicitly choose to combine, separate or ignore states as desired. For example, the designer could combine different forms of walking (fast, slow, pathological), if the sole interest was distinguishing level gait from ascending or descending stairs. This limits the complexity of the classifier by obviating the need to make irrelevant distinctions. However, systems where the states of interest are either large in number or ill-defined may result in classifiers which are either too complex or classify poorly, respectively. Also, while designed for efficiency, the data collection scheme may have difficulty in situations where states are not easily generated (such as environmental monitoring). With knowledge of the time evaluation it may be possible to synthesize this missing data. Otherwise, the available data can be used as a starting point with the assumption that the system will be updated using techniques such as those considered in section 8.3. Finally, while hand-annotation can be time consuming, the use of scripted training sequences should reduce the workload to a manageable level.

5.2 Feature Extraction

Before training the classifier based on the marked examples, a set of features is extracted from the data. These fea-

tures should be both compact and descriptive. For most embedded sensor systems, the data from which we will be extracting features will be in the form of a time series. While it is possible to simply use the values at each point in time, this would be less than robust because of variations both from the structure of the time series and from the noise in the individual data points. To take this variation within single states into account, windowed functions are used to calculate the features.

A set of simple first order functions were chosen as the features — specifically, the windowed mean, variance, minimum and maximum. These features have been used successfully on time series of inertial [2] and video [19] data. They are also mathematically simple, requiring $O(1)$ calculations and $O(n)$ memory (for a window of size n) to update their values at each time step. This simplicity provides two benefits. First, while the classifier training is offline and can therefore be as complicated as necessary, the features must be calculated in real-time and therefore should fall within the limited processing and storage capabilities of embedded microcontrollers. Second, the energy expended to calculate these functions will typically be two or more orders of magnitude less than that necessary to collect the data, allowing us to treat it as negligible.

These are good general statistics for two reasons. Intuitively, the mean is the baseline, the maximum and minimum provide the limits or range and the variance is a measure of energy expended by the subject. Analytically, the mean and variance are the first and second moments (respectively) of a random process. If the window size is chosen to be the period (for cyclic data) or greater than the phenomenon's correlation length (for non-cyclic data), the data stream will be wide sense stationary within any given state, and these values will be constant (with the exception of additive noise). This converts a sequence of time varying values to one which varies with state alone, allowing the use of most supervised classification algorithms.

To extract as many uncorrelated examples from the data stream as possible, a sliding offset of a quarter window length for each example is used (which has been shown to give good results [4]). Also, since power use is correlated to sampling frequency, examples are generated at a number of different rates, allowing the classifier to choose the lowest power example containing the necessary information. The maximum rate for generating examples is the sampling frequency of the training stream and examples are generated at this rate and at power of two divisions thereof (down to some reasonable minimum value).

5.3 Classifier Form

Having collected the data streams and used them to produce training examples, we are now ready to design and construct a classifier to separate the various labelled states. To determine the necessary properties of the classifier, we note that this work seeks to reduce the power usage of sensor nodes through the reduction of sensor usage. Specifically, we seek a collection of hierarchical activation levels to allow the system to make a state determination using as little energy as possible. Hence, the classifier used should be able to make decisions in the same fashion — using more or less data as needed — exploiting the well established training algorithm to produce a classifier which explicitly produces the tiered wake up structure desired.

Therefore, decision trees are used in this framework. Decision trees structure classification in the form of a series of successive queries (usually a threshold on a single feature), with each response leading to a following query until a state is determined [36, Chap. 7.2]. In this way, the tree uses different sets of features to classify different states (or subsets thereof). In the case of an unbalanced tree, some classifications are made with far fewer decisions (and therefore far less energy) than others. Also, the recursive structure makes for a fairly inexpensive classifier suitable for embedded usage, with an average computational complexity of classification of $O(\log n)$ comparisons and a space complexity of $O(n)$(for n training examples) [10, Chap. 8.3]. Overall, the desire for hierarchical activation requires a hierarchical classifier. Similar arguments have been made in both medical [25, Chap. 16] and general [35] contexts.

It is important to note that decision trees can only subdivide the feature space perpendicular to the feature axes, leading to poor performance when a linear (or otherwise) combination of features is necessary to accurately subdivide the space. This limitation can be overcome by careful choice of sensors and their axes of measurement, to create a state space where the single variable cuts made by the classifier are more meaningful. Note that techniques for building decision trees which use combinations of features in their queries have not been particularly successful [7].

5.4 Classifier Implementation

We use the CART decision tree construction algorithms codified by Breiman *et al.* [8]. This technique, known as top-down induction of decision trees, is a simple divide and conquer algorithm which, at each node of the tree, divides the labelled examples in such a way as to maximize a selected criterion. This process continues until each node only contains examples of a single state. Breiman *et al.* proposed the Gini criterion for selecting binary splits of two-class systems:

$$C_{Gini}(s) = \sum_{i \in \text{left,right}} P(i)P(+|i)P(-|i) \qquad (1)$$

where s represents a division of the examples, $P(i)$ is the proportion of the total examples assigned to the child node and $P(+|i)$ and $P(-|i)$ are the proportion of positive and negative examples (respectively) in that node. The split with the smallest value is chosen. Since growing the tree until all of the nodes are pure tends to greatly overfit the data, pruning is performed to find the right-sized tree. We use the technique suggested by the CART algorithms — known as cost-complexity pruning — which selects the tree with the minimal error on a separate testing data set.

The above process is designed to construct the most accurate tree. However, in our case, we are not solely concerned with finding the tree that separates the examples most accurately, but also that does so for the lowest average power. Therefore, we wish to trade off accuracy and power usage in a structured way. We begin by defining the quantity we wish

to affect the growth of the tree:

$$TC = (TC_s + TC_f) \tag{2}$$

where TC is the total test cost of the sensor (in units of power), TC_s is the power used to collect a sample and TC_f is the power used to calculate a single feature based on that sample. For the features chosen, in almost all cases $TC_s \gg TC_f$ and therefore $TC \cong TC_s$. The exception is very low power and passive sensors. Since this is a sequential classifier, once a feature has been used to make a decision, the sensor from which that feature was calculated should then be available to all children of that node for a discounted cost. The use of a sensor which is already being sampled at the desired rate will not require additional energy and therefore should not invoke additional cost ($TC = 0$). An increase in sampling rate should only bear the cost of the increased power usage. Sensors which share a common cost (such as an amplifier) can also be easily accommodated. Any activated sensor that follows one with which it shares a cost is simply discounted by that amount.

To be able to take test cost into account when building the tree, the splitting criterion must be a function thereof. Therefore, we add a generalized multiplicative weighting function:

$$C'(s) = \frac{C(s)}{(\alpha + \beta(TC))^W} \tag{3}$$

The intent is to reorder $C'(s)$, relative to $C(s)$, based on the test cost. Two separate operating ranges are desired. For activated sensors ($TC = 0$), the weighting function should have no effect whatsoever. For unused sensors, the weighting should be proportional to the power usage. W will be used to control the extent of the reordering and α, β are fixed as described below.

We first set $\alpha = 1$ without loss of generality, since it can simply be divided out without altering the ordering of the function. For $TC = 0$, this makes the weighting identically 1 for all W. The value for β is then set based on the specific values of the test cost. Define $\gamma \equiv \beta \min(TC)$. For $\gamma = 10 \gg 1$ and $TC \neq 0$, the α term of the sum can be ignored, and the weighting is proportional to TC as desired.

The parameter W adjusts the relative importance of power in the classifier construction, with $W = 0$ having no effect and $W = \infty$ forcing only the sensor(s) with the lowest test cost to be chosen. Since this is a greedy process, it is most likely that the effect of this parameter will not be smooth, but rather that a range of values will all result in identical trees being grown. As long as the total test cost for classification is roughly monotonically non-decreasing with increasing W and the number of different trees is non-degenerate, this stepwise behaviour is not of particular concern. The overall goal is simply to be able to grow a population of different trees by varying W. The exact relationship between W and total test cost is not important.

In [5], it is shown, using three standard data sets, that this test cost weighting achieves the goals described above and creates a population of classifiers at different points in the power/accuracy plane. We quickly note two results. First, the most accurate trees were often not those grown with $W = 0$. Second, the depth of the tree (roughly equivalent to the evaluation latency) peaked for middling values of W. The reader is directed to the above work for further details.

Several other weighting function have been used in the literature. The functions of Tan [33] and Norton [26]:

$$C'(s) = \frac{[C(s)]^2}{TC} \text{ and } C'(s) = \frac{C(s)}{TC}$$

are equivalent to ours for $\alpha = 0$, $\beta = 1$ and $W = 0.5$ and 1, respectively. Neither function allows sensor costs to be set to zero in the case of repeated use of the feature as this would make $C'(s)$ undefined. Further, their choice of parameters (α, β, W) are not justified beyond unsupported claims of optimality and W cannot be varied to alter tree construction. Núñez [27] models the system as a Shannon transmission line [30] and finds a signal to noise ratio of:

$$C'(s) = \frac{2^{C(s)} - 1}{(1 + TC)^W}$$

where $C(s)$ is the entropy of the split and $TC \gg 1$. However, since $0 < C(s) < 1$, it is difficult to gauge the similarity between this form and our own. Costs are allowed to be zero, though Núñez does not consider the case of costs which vary within the tree.

6 Embedded Implementation

6.1 Overview

Algorithm 1: Overview of Program Flow

```
1  while true do
2  |  Collect data;
3  |  Run classifier;
4  |  if classifier returns answer then
5  |  |  Execute desired response;
6  |  else
7  |  |  Turn on necessary sensor;
8  |  end
9  |  Turn off unnecessary sensors;
10 |  Sleep until next cycle;
11 end
```

The basic flow of the embedded code is shown in algorithm 1. This loop is repeated as long as the node is active, based on the assumption that the main purpose of the system is to determine the current state and respond to it.

Data collection is determined by the instantaneous activity level of the system (specified by the last non-leaf node encountered in the decision tree), which selects the sensors to activate and sets their sampling rate. In most cases, the sensors are powered down between cycles to save energy and therefore need to be activated before a sample can be taken. To a first order, the processor wakes up and activates the appropriate sensors. Once their output becomes valid, the processor collects the data (almost always using the ADC) and then turns them off. The main constraint is that the turn on time of the sensors be shorter than the cycle time of the system. If violated, it is necessary to leave the sensors on continuously, which reduces the achievable power savings.

Given the sensor data, it is now possible to attempt to determine the current system state. The evaluation of the clas-

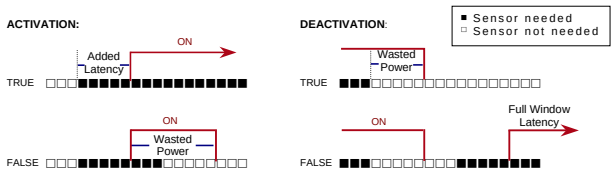

Figure 3. Latency and wasted power for sensor activation and deactivation scenarios

sification tree itself is straight-forward. The process is simply a series of comparisons of a feature calculated from a window of sensor data to a fixed threshold, with each result either determining the next test or returning the state. If the returned state has a response associated with it by the application designer (*e.g.* collecting data, or cuing the user), it is executed at this point. Tree evaluation can fail when the data necessary to determine the features are not available, either because the sensor is currently inactive or has not been active long enough to the fill the data window (since this is a tiered wakeup system, this may not be a rare occurrence). In this circumstance, the only solution is to activate the sensors and wait for one window length to be able to proceed. During this time, the system is said to be in an indeterminate state, and no responses are executed. Separately, the sensors which were active but were not used in the decision process are deactivated. In both cases, sensor noise can lead spurious de/activation requests by sending the tree evaluation along the incorrect path. Adding some hysteresis can help alleviate this problem, and is discussed in below.

Finally, the system is put to sleep until the next cycle. The processor is set to awaken the next time data collection is required, which is determined by the update rate of the active sensors. The only components active during this phase should be the sensors that are not duty cycled.

6.2 Power Cycling and Hysteresis

A key implementation issue to consider is that of noise, specifically with respect to sensor de/activation. The concern is simple — that a spurious transition to a higher/lower node in the tree (*i.e.* lower/higher activity level) from sensor or sampling noise not result in the needless de/activation of a sensor. Therefore, it is required that the de/activation of a sensor be requested (by arriving at a tree node which does/doesn't use it) some fixed number of times before the state change takes place. While the problem appears symmetric, this is not the case. Because of the windowed nature of the features calculated, a newly activated sensor must be on for the full length of the data window before it can be used. The cost of accidentally turning off a sensor is the loss of all of that data and therefore results in a large latency while the delay caused by waiting for confirmation results in only a small amount of wasted power. By contrast, the cost of accidentally turning on a sensor is the power wasted until the mistake can be corrected, while the waiting for confirmation only causes a slight increase in latency (especially compared to the length of the data window). The two cases are shown in Figure 3, where ■ and □ represent the need for a sensor and the lack thereof, respectively. The de/activate markers are generated for a threshold of five requests in this figure.

It is apparent that the system should be quick to activate a sensor but loathe to turn one off. The on latency will therefore be set rather low, on the order of one tenth of the window size. The off latency, by contrast, will should be much higher, on the order of a half of the window size.

Smoothing of the output state is also desired to avoid spurious execution of the response function, which could increase power usage and reduce the utility of the system. This smoothing will take the form of hysteresis, where the system requires that the tree output the same state for a fixed number of cycles in a row before a change is acknowledged. This will be in the range of one quarter of the window size.

7 Testing and Analysis

7.1 Test Scenario

	Power usage (μW) at			
	25 Hz	50 Hz	100 Hz	200 Hz
ADXL202	1031	2031	4031	4830
ADXRS300	22343	24831	24831	24831
ENC03J	11931	11931	11931	11931
Tilt	5.85	11.7	23.4	46.8

Table 2. Power usage of sensors by sampling rate.

As a detailed testing scenario for this framework, we decided to mimic the wearable gait laboratory detailed in [3]. A six degree-of-freedom inertial module was built using two Analog Devices ADXL202 accelerometers, two Murata ENC03J gyroscopes and a single Analog Devices ADXRS300 gyroscope. A four-way static tilt switch provided for additional micropower single-bit acceleration measurement and a Texas Instruments MSP430F1611 microprocessor was used to collect and process the data. The hardware was attached to the heel of the user's shoe with a thermoformed attachment. Data was collected at 200 Hz and versions of the data streams for sampling rates of 25, 50 and 100 Hz calculated using the `downsample` function in MATLAB (which does not apply an anti-aliasing filter). A window size of 1.2 sec was chosen based on the collected data. The power usage (including feature calculation) of the sensors at various sampling rates is shown in Table 2. Shaded cells indicate that the sensors cannot be power-cycled at that rate. Note that the energy use of the ADXL202 includes a buffer to reduce the input impedance to the microprocessor's ADC. Also, because the ADXL202 is a dual-axis package, the test costs of the y-axis (upward) and z-axis (lateral) accelerometer are combined, *i.e.* use of one makes the other available for zero cost.

Our goal was to descriminate between wide variety of different ambulatory activities. The set of activities chosen was: normal level gait, walking uphill, walking downhill, ascending stairs and descending stairs. A shuffling motion was also recorded, where the user specifically attempted to mimic the shuffling motion of a patient with Parkinson's Disease (PD) [12]. This data set allows us to create classifiers which attempt to separate a single (complex) ambulatory activity from the rest. It is often the case that only one of these motions is of interest for a given patient. When treating PD patients, a doctor would be most interested in collecting information about the frequency of and parameters describing the patient's shuffling episodes [40]. For patients with total

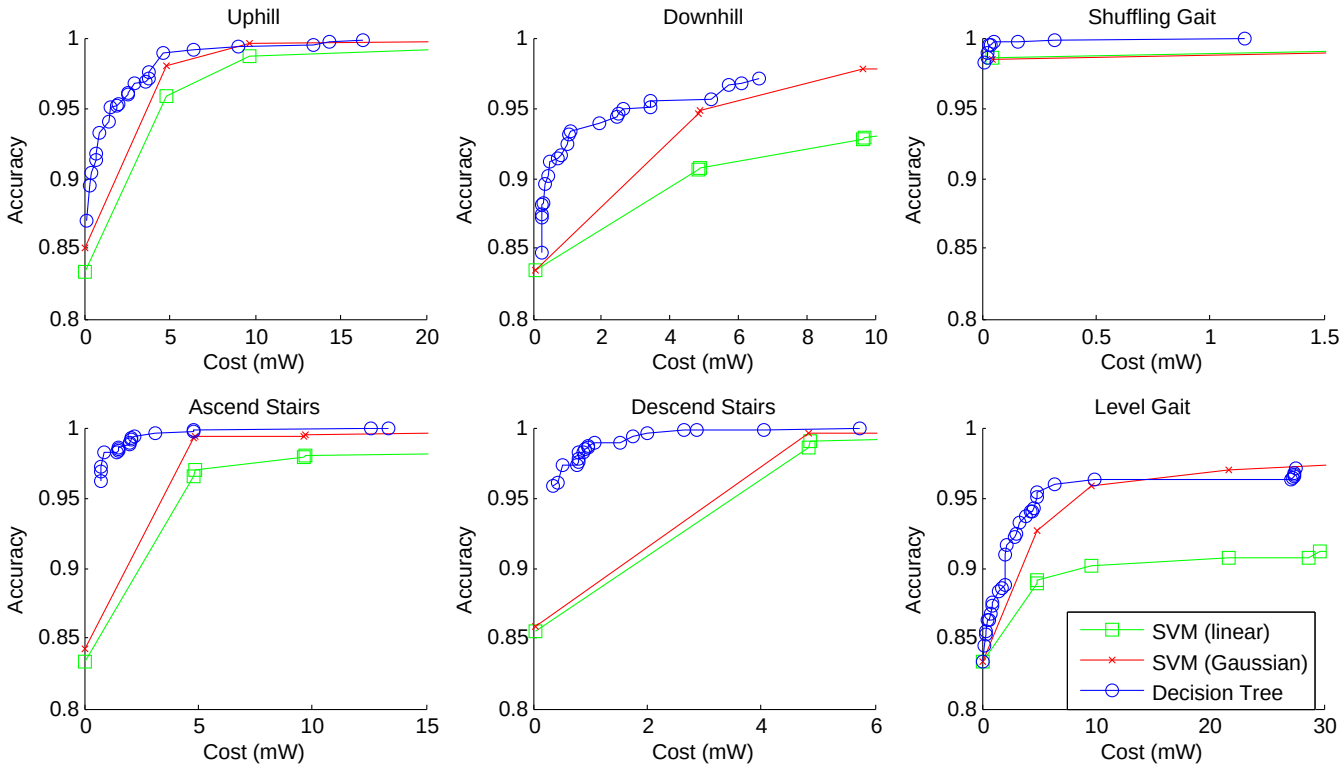

Figure 4. Power/accuracy tradeoff for decision trees and SVM

knee replacement, activities such as ascending stairs (where the knee flexion is $> 90°$) are the most important to measure [18]. Both of these cases call for far richer classifiers than those used to simply separate ambulatory from non-ambulatory (roughly: still) states.

To represent the variability of user motion, training data was collected on five separate days, each time at a different point in the user's waking cycle. The active data streams contained a segment of at least 2 minutes in length of each of the motions. These motions were collected individually and in isolation, since only the states (and not the transitions between them) were of interest. Annotations were later added to the data stream based on marks in the data stream added by a user-controlled pushbutton.

Several days after the last active data stream was taken, a further data stream was collected for the purposes of simulating the real-time operation of the classifier. In the course of a single continuous session, the wearer performed two segments (roughly 10-15 seconds each) of each of the motions of interest. The total running time of this stream is 200 seconds. The data from this stream was not used to either grow, prune or test the classifier, allowing for an independent real-time simulation.

7.2 Static Classifier Performance

Classifiers were trained for the six different cases of one of the ambulatory motions being the positive class and the others being grouped together as the negative class. Each classifier will therefore attempt to separate a single type of motion from the other five. This provides a fairly complex task, such that power savings will be achieved from the appropriate selection of sensors to sample, rather than through long periods of idling. To determine how the classifier responds to the availability of features calculated at different sampling rates, the following combinations were tested: all the frequencies together, each frequency individually and the pairwise combination of adjoining rates.

The classification trees were trained using a two-part holdout procedure. The full data set was divided into 5 segments, with one acting as the holdout test set and the other four used as the training set. The portion used as the test set was rotated, such that all five segments were eventually used. A 20% holdout pruning set (from the four segments which make up the training set) was treated in the same fashion. Overall, 25 classifiers were built for each of ten values of W between 0 and 0.29 — the maximum useful value of W [5]. Accuracy and average test cost were found using the test set.

Support vector machines [36, Chap. 5.4] were trained for the same tasks as above to act as a point of comparison. Both Gaussian and linear kernels were used, and the accuracy was estimated from 5 loops of 5-fold cross-validation to mimic the same training set/test set combinations used for the decision trees. While decision trees select the useful features as a fundamental part of their operation, SVMs use all of the features provided in the training set. Therefore, to calculate the power/accuracy trade-off of the SVMs, classifiers were trained for all 63 possible combinations of active sensors run at the maximum sampling rate.

The results are plotted in the power/accuracy plane, with each classifier being a single point (Figure 4). An algorithm is considered superior to another if its curve is closer to the

	Gaussian SVM		This Framework		Ratio (SVM:This)	
	Accuracy	Power (mW)	Accuracy	Power (mW)	Accuracy	Power
Level Gait	0.9702	21.59	0.9601	6.37	1.011	3.39
Uphill	0.9953	9.66	0.9888	4.66	1.006	2.07
Downhill	0.9781	9.66	0.9667	5.70	1.011	1.69
Ascend Stairs	0.9921	4.83	0.9912	2.06	1.001	2.35
Descend Stairs	0.9959	4.83	0.9890	1.07	1.006	4.52
Shuffling Gait	0.9997	4.83	0.9984	0.32	1.001	15.1

Table 3. Comparison of best practises classifiers to this framework

top left corner (high accuracy/low power). The curves for each algorithm are simplified by showing only the points which make up the non-decreasing hull of the data, since any points below this curve are sub-optimal. The power axis is set such that the whole curve for the decision trees is visible as well as at least two points of the SVM curves. While different curves are made up of a different number of points, this is a function of the classifiers generated in each case and does not hold any intrinsic meaning.

The first step is to compare the performance of the decision tree classifiers with those obtained using support vector machines. In all cases the decision trees, over their range of operation, are superior to the linear SVM, achieving the same accuracy for less power. They are superior to the Gaussian SVM in all cases but level gait. Further, note that the Gaussian SVM is unsuitable for implementation in an embedded platform. While a linear SVM requires a dot product of vectors with length equal to the number of features, a Gaussian SVM requires one such product for each support vector (*e.g.* ~ 100 for walking). This is not only time consuming, but requires large amounts of static memory. Thus, this comparison is relevant numerically but impossible to implement practically.

Surprisingly, training with all available sampling rates did not give the best decision trees in any of the cases. Rather, the tasks fall into two sets. For uphill, downhill and level gait, the 200 Hz data alone seems, in general, to give the best classification results. For ascending and descending stairs and the shuffling gait, the 25 Hz data alone is superior. These motions appear to be simpler and more structured than the first three mentioned above, and therefore can be differentiated with less data and thus less power.

More importantly, in all cases there is a knee in the power/accuracy curve for the decision trees, after which there is minimal gain in accuracy for a large increase in power use. Prior to this point, the accuracy of the decision tree is improved by adding more sensors. Afterward, the accuracy is improved by moving decisions involving the more expensive sensors closer to the root node. This strongly supports the claim that a hierarchical activation system can provide strong classifiers for reduced cost.

To demonstrate this point, Table 3 compares results of the current practises used for training SVMs with the framework presented here. Feature selection is used for the SVMs, with the features calculated at the maximum sampling rate — the standard practise, at least in the case of embedded sensors. For each algorithm, the best classifier is defined as the one with the lowest power usage amongst those with accuracies within 1% of the most accurate classifier. For our framework, this corresponds to the knee point defined above. This definition was chosen to avoid selecting very power hungry classifiers which are only marginally more accurate than those with more reasonable power usage.

In this comparison with SVMs, our framework performs quite well. SVMs do offer an accuracy improvement on the order of 0.5%, though at a power increase of about three times in most cases. It appears that the hierarchical activation of sensors allows our framework to compete with a classifier of greater descriptive power.

7.3 Real-time Classifier Performance

Two concerns arise when using trees trained with static data in a real-time system. The first is the effect of the latency inherent in activating sensors and whether this will lead to missed detections. The second is the issue of the generalizability of the classifier itself — *i.e.* did it overtrain to the sample set. Testing was done using the trees at the knee points of the power/accuracy curves.

A few key facts about these trees are noted. First, as mentioned above, they use only a single sampling rate throughout their operation. Second, the trees contain a large number of leaf nodes with populations of much less than 1% of the data set. These leaves are very expensive to calculate and provide only a tiny improvement in accuracy and hence were pruned. Notably, these pruned trees used three sensors exclusively: the tilt switch and the y- and z-axis accelerometer. Since the latter two are contained within a single sensor package, there were only two activation levels in our trees (with the exception of the shuffle classifier, which has only one). This limits the possible benefits of our system and suggests that a different test application would allow for greater power savings.

	% Time at High Activation Level		
	Static	Simulation	Embedded
Level Gait	74	95	95
Uphill	63	72	86
Downhill	71	85	78
Ascend Stairs	74	88	98
Descend Stairs	38	63	47
Shuffling Gait	100	100	100

Table 4. Percentage of time in higher activation levels

Each classifier was both simulated on the continuous stream set aside for this purpose and also tested in our embedded sensor node. Table 4 shows the percentage of time during which the tree was in the higher activation level for the static (with all sensors instantly toggled) and real-time (with sensor on/off latency as above) trees. The latter includes both simulations and tests run on the embed-

ded implementation. We assume that a binary wakeup system would have been triggered by any motion, and therefore would have a value of 100% for each task. The static tree is in the high energy state far less often than the real-time systems. There are two main reasons for this. First, the turn off hysteresis led to unnecessary sensors being powered. Reducing this value can save power, though it risks increasing the latency and causing missed events. Second, the structure of the tree is such that examples from the negative states can fall into leaves at a number of different levels of the tree. It is surmised that during the testing of the embedded node, the user's gait was such that the deeper leaves were needed, thereby increasing the power usage. It is possible that the classifier would use less power in further tests.

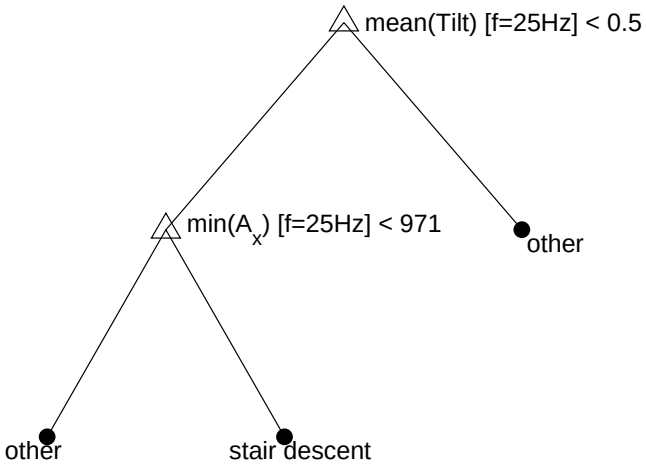

Figure 5. Classification tree for stair descent

The results for the stair descent tree (Figure 5) are examined in greater depth. The left-hand column of plots in Figure 6 shows the ground truth of the user annotation and four different interpretations of the operation of the classifier. The first is the state output of the static classifier and the second shows the state output for the real-time classifier. Note that, by definition, the real-time classifier cannot respond more quickly. The third plot is the number of sensors active at any given point and the last shows the power consumption of the active and awakening sensors. Just as the state output of the real-time classifier is a subset of the state output of the static classifier, it is also a subset of the activation level, which is a subset of the power usage. The right-hand column shows the same information, this time collected from the system running in the testing of the embedded node.

While quantitative evaluation of false positives and negatives is not appropriate here because of the conservative nature of the annotation and the differences between the individual test runs, qualitatively we note that the stair descent classifier detected the last two-thirds of each decent and had only minor blips otherwise. As mentioned above, the hysteresis was a major source of wasted energy, as the classifier drew the power associated with the higher activation level far more often than it was actually in it. There is a baseline power usage (roughly 4 mW) associated with execution of the classification tree (and associated bookkeeping) in the microcontroller. This could be reduced through software or hardware redesign for increased efficiency. Nonetheless, our system is concerned with the difference in sensor power usage, and this was as predicted.

The other classifiers fall into two categories. The stair ascent and shuffle classifiers, which like the stair descent classifier run at 25 Hz, both correctly recognize virtually all of the associated motions while producing only minimal false negatives (and then only during state transitions). The downhill, uphill and level gait classifiers, which run at 200 Hz, do not fare so well. These tasks are more difficult, primarily because the three motions themselves are difficult to disambiguate from each other. In each case, the classifier recognizes approximately two-thirds of the associated motion and tends to flicker on and off only during the other two motion (net total of a quarter of the time). While these results may seem numerically disappointing, they are no worse than those of the SVMs.

The operation of the classifier during transitions between states is ambiguous, since portions of the transitions include shallow ascents and short periods of level gait. Since these are not annotated, clear conclusions cannot be drawn, though it appears that the classifiers are not limited solely to the parameters (such as grade of slope) on which they were trained.

Overall, the classifiers were successfully implemented in hardware and performed as well as their static counterparts. The latency does not appear to have lead to missed detections, though the hysteresis — which was meant to counteract the latency — instead led to increased power usage and should be reduced in the future. The classifiers do not appear to have overgeneralized, though the natural variation in human motion caused the power usage to differ from the predicted values in the tested cases. Further study is necessary to determine if it is possible to build trees which will give more consistent power usage over their range of operation.

8 Future Work

8.1 Improvements to the Implementation

As it is the core of this framework, we concentrate our effort on improving the classification tree construction algorithms and insuring that the chosen problems can benefit from this approach.

Broadening the choice of features should allow for more compact and accurate trees. The current set of features used to train the classifier were chosen based on their simplicity and generality. However, for any particular problem, it is likely that the application designer knows of (or can find in the literature) other features which are useful within the specific domain. It should be possible for such features to be calculated from the training set and used in the classifier with a minimum of hassle for the designer. While it will often be the case that these features are more computationally expensive than the current set, the selected processor should be sufficient in most cases. Similarly, it should be possible to apply knowledge of sensors which, when used in combination, more effectively partition the data set. The current discounting of test cost can handle such features in all orderings (*i.e.*, regardless of whether the combination or the individual sensors are used first). However, since the classifier does not have the ability to adjust the parameters of the

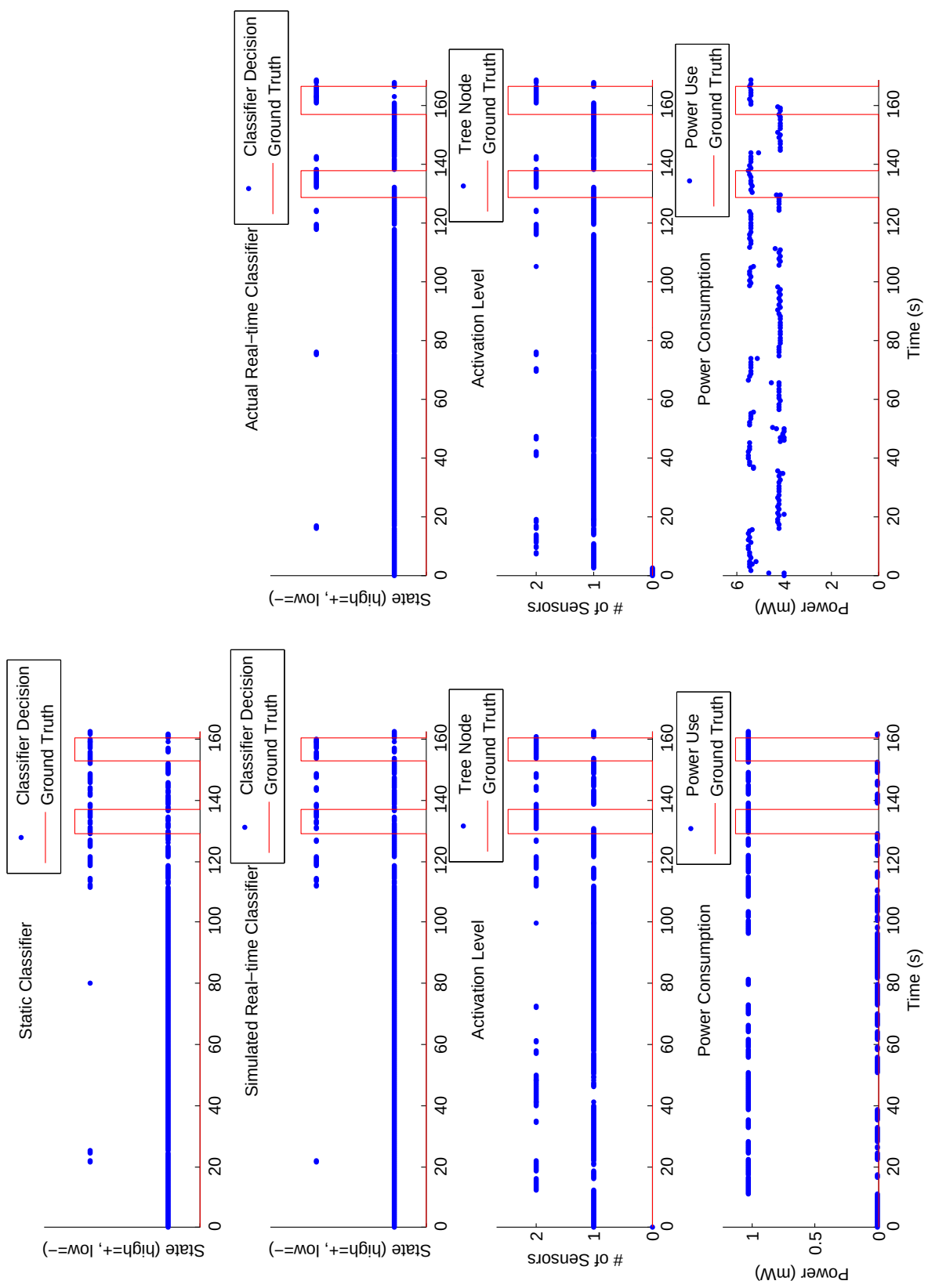

Figure 6. Simulation (left) and embedded operation (right) of stair descent classifier

features, the exact combination must be fixed ahead of time.

The testing application itself needs to be reconsidered. In our case, the two-axis accelerometer was too useful in and of itself, and the gyroscopes were so expensive that they were never included by the tree construction algorithm, even at the lower branches. An application needing a greater variety of sensors measuring significantly different modalities would provide a much better test. The effect of the range (*i.e.* same or multiple orders of magnitude) of sensor costs should also be examined. Finally, this work suggests that compound sensors which allow individual control of both the activation and accuracy of their individual components could allow for the construction of more efficient sensor nodes.

8.2 Extension to Sensor Networks

We are considering modifications to extend this work to networked sensor systems, thereby allowing for larger and more complex applications. Overall, the goal is to exploit the potential benefits to the network as a whole from nodes informing their neighbours of their current sensing and/or state, thus allowing them to adjust their own sensing to guarantee that the network (rather than the individual nodes) makes decisions in the most efficient fashion. While most sensor networks fuse data to get the best possible result, our goal is simply to collect enough information to make a decision on a tree node. Sensor nodes can communicate either their current measurement(s) and/or state. These can be transferred between the nodes in either a peer-to-peer or centralized format, dependant on whether the information is relevant globally or only locally.

Structurally, data from external sources would not be handled any differently than local data by the decision tree training algorithms. Their test cost would be the marginal power usage of reception beyond the communication already necessary to the network. This data is most likely to be used if the test cost is small, such as in networks which communicate continuously (*e.g.* to maintain synchronicity of timers), for data from high power sensors (*e.g.* sonar), or when they are the only available source. Also note that, for the case of homogenous nodes, the same classifier can be run on all nodes of the network.

We examine the specific case of tracking problems is depth. In these applications, the individual nodes will switch states as the phenomenon moves through the sensed area. Their measurements and states will be correlated over time based on the trajectory and speed of the phenomenon, such that the values and state of one node will contain information about the current and future state of its neighbours. Therefore, energy can be saved by gaining state knowledge from other nodes rather than through sensing. Peer-to-peer communication is most appropriate, as only neighbouring nodes can benefit from the information. At this point, there are two cases based on the speed of the phenomenon relative to the node spacing. If the phenomenon is moving slowly, then it is most efficient to transmit sensor data, which each node will use to determine not only whether the object is present, but if it is the closest node to it. This is similar to work by Zhao *et al.* [39], where nodes in a peer-to-peer network follow a data-accuracy gradient to determine which node should sample a static phenomena. If the phenomenon is moving quickly, then it is move valuable for nodes to transmit their states. A message from a neighbour stating that it has detected an object would become part of the tiered wakeup procedure, allowing the node to transition to a higher activity level in anticipation. This should greatly reduce the latency and increase the amount of interesting data captured, while an isolated node may well miss the phenomenon entirely. Similarly, a message from a neighbour stating that the object is now out of range would likely move a node down into a lower activity level. Note that while a centralized approach would have the benefit of more accurate trajectory prediction (through knowledge of the network topology and track history), it would require substantially more communication. He *et al.* [14] examine this problem for the case of a two-tier network — a fixed set of vigilant (always-on) sentries which wake the other nodes when triggered — and achieves a significant increase in network lifetime.

8.3 Unsupervised Online Training

Unsupervised online training of these sensor nodes would have two main benefits. The first is the ability to detect rare or unexpected states which might be missed through the training process. For a gait monitoring system, tripping is an example of an event which is unlikely to be in a sample data stream (almost regardless of length), could possibly be missed by the designer and would be of great value to detect. The second benefit would be the potential of the system to alter the classifier (by adjusting constants or adding states) over time as the environment or user changes (*e.g.* the appearance of a shuffling gait). While more general and less taxing on the designer, it should be noted that unsupervised training has the drawback of creating unlabelled states, making both real-time and offline analysis more difficult because of the lack of context.

To allow the system itself to remain relatively low power, heuristics for the occurrence of new or interesting events will need to be created. Starting from a supervised solution, one potential method is to look for, on a micro level, state thrashing in the decision tree or, on a macro level, a large divergence from the expected power usage. While these conditions can possibly be due to poor training, they could also indicate that the system has an emergent condition. A complete data stream can be collected for analysis and comparison to known states. If the differences are minor, this suggests a change in state statistics over time. Adjustment of the splitting threshold of the thrashing decision node should correct this problem. If there is a significantly difference, a new state is added. This would require retraining of the decision tree. Examples of each state could be collected during regular operation. While training a decision tree is time intensive, it can still be accomplished *in situ* with the current microprocessor as no floating point operations are required (for $W = 0$). However, an offline process would still be required to construct a large enough population of trees for power/accuracy optimization.

As a further step to possibly capture very short states, full data stream segments can be randomly collected throughout the day at times when the device would otherwise be in a low-power state (*i.e.* when supposedly nothing interesting is taking place). Information theoretic techniques can be used

as a low-level assessment of the complexity, and therefore potential interest, of the data, with a strong enough result leading to a new state being added to the decision tree. Retraining, as above, would then be required. This method will take a very long time to accidentally catch one of these events and it can be thought of as spreading out the power usage of long term continuous data capture across a large number of battery cycles, such that no single cycle is noticeably shortened.

Finally, given advance knowledge of possible circumstances which would drastically alter the state detection (*e.g.*, for the wearable gait lab, a sharp increase in outdoor temperature might correlate to reduced activity levels), low-power static sensors with preset thresholds can be added to detect these cases for a nominal reduction in battery life (see [24] for one such system). Because of the compactness of their representation, replacement decision trees could be stored locally for use in such an eventuality.

9 Conclusions

We have presented a three-component framework for power-efficient detection in wearable sensors. The first is modular hardware platform for ease of application prototyping which has been adapted to better reflect low-power goals. The second and key component is a semi-autonomous classifier construction algorithm. Since the specific goal of this work is to create a hierarchy of activation levels to allow the system to make a state determination as efficiently as possible, decision tree classification was used. By structuring classification as a series of successive queries, the tree uses different sets of features to classify various states, with some requiring far fewer decisions, and therefore far less energy, than others. The standard top down induction algorithm for decision trees was modified by weighting the splitting criterion by the energy cost necessary to collect the sensor data used to calculate the features. As appropriate, this energy cost is discounted based on prior use of a sensor in the tree. The weighting is parameterized, and allows for the construction of a collection of trees at various points on the power/accuracy curve. The final component is a design for an embedded implementation of this classifier for use with wearable sensor nodes. This implementation takes into account a number of issues deriving from the real-time sequential nature of the system, including smoothing the state output and requests for de/activation of sensors to avoid expensive spurious actions.

An application based on a wearable gait monitor provides quantitative support for this framework. The decision tree classifiers achieved roughly identical detection accuracies to those obtained using support vector machines while drawing three times less power. Both simulation and real-time operation of the classifiers demonstrate that our multi-tiered classifier determines states as accurately as a single-trigger (binary) wakeup system while drawing as little as half as much power and with only a negligible increase in latency.

10 Acknowledgements

The authors thank Michael Lapinski for his invaluable efforts in helping to implement the embedded code and test application for this framework and Mathew Laibowitz for his help with the IMU redesign. We would also like to thank the anonymous reviewers and our shepherd for fruitful discussions. The authors also acknowledge the support of the Things that Think Consortium, as well as all the sponsors of the MIT Media Lab.

11 References

[1] A. A. Abidi, G. J. Pottie, and W. J. Kaiser. Power-conscious design of wireless circuits and systems. *Proceedings of the IEEE*, 88(10):1528–1545, October 2000.

[2] K. Aminian, P. Robert, E. E. Buchser, B. Rutschmann, D. Hayoz, and M. Depairon. Physical activity monitoring based on accelerometry: validation and comparison with video observation. *Med Biol Eng Comput*, 37(3):304–308, May 1999.

[3] S. J. M. Bamberg, A. Y. Benbasat, D. M. Scarborough, D. E. Krebs, and J. A. Paradiso. Gait analysis using a shoe-integrated wireless sensor system. *IEEE Transactions on Information Technology in Biomedicine*, 2007. *To appear.*

[4] L. Bao and S. S. Intille. Activity recognition from user-annotated acceleration data. In *Proceedings of PERVASIVE '04*, LNCS 3001, pages 1–17. 2004.

[5] A. Y. Benbasat. *An Automated Framework for Power-Efficient Detection in Embedded Sensor Systems*. PhD thesis, Program in Media Arts and Sciences, Massachusetts Institute of Technology, February 2007.

[6] A. Y. Benbasat and J. A. Paradiso. A compact modular wireless sensor platform. In *Proceedings of IPSN '05*, pages 410–415. 2005.

[7] L. Breiman and J. H. Friedman. Tree-structured classification via generalized discriminant analysis: Comment. *Journal of the American Statistical Association*, 83(403):725–727, Sept. 1988.

[8] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.

[9] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proceedings of the VLDB '04*, pages 588–599. 2004.

[10] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley, 2nd edition, 2001.

[11] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler. Design of a wireless sensor network platform for detecting rare, random and ephemeral events. In *Proceedings of IPSN '05*, pages 497–502. 2005.

[12] GaitBrowser: Pathological Gait Viewer. http://guardian.curtin.edu:16080/cga/GaitBrowser/.

[13] A. Gelb, ed. *Applied Optimal Estimation*. MIT Press, 1974.

[14] T. He, B. Krogh, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, and J. Hui. Energy-efficient surveillance system using wireless sensor networks. In *Proceedings of Mobisys '04*, pages 270–283. 2004.

[15] S. S. Intille. Designing a home of the future. *IEEE Pervasive Computing*, 1(2):76–82, April 2002.

[16] S. S. Intille, L. Bao, E. Munguia Tapia, and J. Rondoni. Acquiring in situ training data for context-aware ubiquitous computing applications. In *Proceedings of CHI '04*, pages 1–8. 2004.

[17] A. Jain and E. Chang. Adaptive sampling for sensor networks. In *Proceedings of the First Workshop on Data Management for Sensor Networks*, pages 10–16. 2004.

[18] D. E. Krebs, J. I. Huddleston, D. Goldvasser, D. M. Scarborough, W. H. Harris, and H. Malchau. Biomotion community-wearable human activity monitor: Total knee replacement and healthy control subjects. In *Proceedings of BSN '06*, pages 109–112. 03-05 April 2006.

[19] L. Lee. *Gait Analysis for Classification*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 2002.

[20] J. Lester, T. Choudhury, and G. Borriello. A practical approach to recognizing physical activities. In *Proceedings of Pervasive '06*, pages 1–16. 2006.

[21] J. Lifton, M. Feldmeier, Y. Ono, C. Lewis, and J. A. Paradiso. A platform for ubiquitous sensor deployment in occupational and domestic environments. In *Proceedings of SPOTS '07*, pages 119–127. 2007.

[22] H. Liu, A. Chandra, and J. Srivastava. eSENSE: energy efficient stochastic sensing framework for wireless sensor platforms. In *Proceedings of IPSN '06*, pages 235–242. 2006.

[23] C. Luschi, M. Sandel, P. Strauch, J. J. Wu, C. Ilas, P-W. Ong, R. Baeriswyl, F. Battaglia, S. Karageorgis, and R-H. Yan. Advanced signal-processing algorithms for energy-efficient wireless communications. *Proceedings of the IEEE*, 88(10):1633–1650, October 2000.

[24] M. Malinowski, M. Moskwa, M. Feldmeier, M. Laibowitz, and J. A. Paradiso. Cargonet: A low-cost micropower sensor node exploiting quasi-passive wakeup for adaptive asychronous monitoring of exceptional events. In *Proceedings of SenSys '07*. 2007.

[25] R. S. Michalski, I. Bratko, and A. Bratko. *Machine Learning and Data Mining: Methods and Applications*. John Wiley & Sons, Inc. New York, NY, USA, 1998.

[26] S. W. Norton. Generating better decision trees. In *Proceedings of the Eleventh International Conference on Artificial Intelligence*, pages 800–805. 1989.

[27] M. Núñez. The use of background knowledge in decision tree induction. *Machine Learning*, 6(3):231–250, May 1991.

[28] V. Raghunathan, S. Ganeriwal, and M. B. Srivastava. Emerging techniques for long lived wireless sensor networks. *IEEE Communications Magazine*, 44(4):108–114, April 2006.

[29] M. Rahimi, R. Pon, W. J. Kaiser, G. S. Sukhatme, D. Estrin, and M. B. Srivastava. Adaptive sampling for environmental robotics. In *Proceedings of ICRA '04*, pages 3537–3544. 2004.

[30] C. E. Shannon and W. Weaver. *Mathematical Theory of Communication*. University of Illinois Press, 1963.

[31] A. Sinha, A. Wang, and A. Chandrakasan. Algorithmic transforms for efficient energy scalable computation. In *Proceedings of ISLPED '00*, pages 31–36. 2000.

[32] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. In *Proceedings of SenSys '04*, pages 214–226. 2004.

[33] M. Tan. Cost-sensitive learning of classification knowledge and its applications in robotics. *Machine Learning*, 13(1):7–33, October 1993.

[34] G. Tolle, J. Polastre, R. Szewczyk, N. Turner, K. Tu, P. Buonadonna, S. Burgess, D. Gay, W. Hong, T. Dawson, and D. Culler. A macroscope in the redwoods. In *Proceedings of SenSys '05*, pages 51–63. 2005.

[35] P. Turney. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, 2:369–409, 1995.

[36] A. Webb. *Statistical Pattern Recognition*. Wiley, New York, 2nd edition, 2002.

[37] L. Yu and A. Ephremides. Detection performance and energy efficiency of sequential detection in a sensor network. In *Proceedings of HICSS '06*, volume 9. 2006.

[38] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi. Hardware design experiences in ZebraNet. In *Proceedings of SenSys '04*, pages 227–238. 2004.

[39] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration. *IEEE Signal Processing Magazine*, pages 61–72, March 2002.

[40] H. Zhu, J. J. Wertsch, G. F. Harris, J. D. Loftsgaarden, and M. B. Price. Foot pressure distribution during walking and shuffling. *Arch Phys Med Rehabil*, 72:390–397, May 1991.