An Automated Framework for Power-Efficient Detection in Embedded Sensor Systems

by

Ari Yosef Benbasat

S.M., Massachusetts Institute of Technology (2000)
B.A.Sc., University of British Columbia (1998)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2007

Author ____
Program in Media Arts and Sciences
February 9, 2007

Certified by ____
Joseph A. Paradiso
Associate Professor of Media Arts and Sciences
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by ____
Andrew B. Lippman
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

An Automated Framework for Power-Efficient Detection in Embedded Sensor Systems

by

Ari Yosef Benbasat

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on February 9, 2007, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Media Arts and Sciences

Abstract

The availability of miniature low-cost sensors has allowed for the capture of rich, multimodal data streams in compact embedded sensor nodes. These devices have the capacity to radically improve the quality and amount of data available in such diverse applications as detecting degenerative diseases, monitoring remote regions, and tracking the state of smart assets as they traverse the supply chain. However, current implementations of these applications suffer from short lifespans due to high sensor energy use and limited battery size. By concentrating our design efforts on the sensors themselves, it is possible to construct embedded systems that achieve their goal(s) while drawing significantly less power. This will increase their lifespan, allowing many more applications to make the transition from laboratory to marketplace and thereby benefit a much wider population.

This dissertation presents an automated framework for power-efficient detection in embedded sensor systems. The core of this framework is a decision tree classifier that dynamically orders the activation and adjusts the sampling rate of the sensors, 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 fashion. Use of a sensor set which measures the phenomena of interest in multiple modalities and at various rates further improves the power savings by increasing the information available to the classification process.

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

Thesis Supervisor: Joseph A. Paradiso
Title: Associate Professor of Media Arts and Sciences, Program in Media Arts and Sciences

An Automated Framework for Power-Efficient Detection in Embedded Sensor Systems

by

Ari Yosef Benbasat

The following people served as readers for this thesis:

Thesis Reader____

Rosalind W. Picard
Professor of Media Arts and Sciences
Program in Media Arts and Sciences

Thesis Reader____

Mani B. Srivastava
Professor of Electrical Engineering
University of California, Los Angeles

Acknowledgments

Some titles, by their very nature and long history, carry with them certain meaning. We know that Generals lead tens of thousand with courage, Rabbis preach the word of God with humility, and Knights face their challenges with grace. But such is not the case for the title of Doctor of Philosophy. It remains to most an unknown, an enigma. The title carries with it only what the public sees in their few encounters with us. And thus it behooves us to show that this title does not represent what we have learned, but instead what that hard earned knowledge has given us: a measure of courage, a dose of humility, and a glimmer of grace.

Joe Paradiso has been instrumental in ever aspect of my career, and my committee members, Rosalind Picard and Mani Srivastava, were invaluable. To them, and the many other friends and colleagues who have helped me get to this point, I make this promise: I will cherish the lessons you have taught me, and the example you have set. I will remember that I have much still to learn, and that this is a blessing. And I promise to set an example to those who follow, so that one day they may lead.

Cambridge, MA
February 2007

Contents

List of Figures

List of Tables

Notation

To simplify references to various specifications of the hardware, we adopt the following structure:

$$\textsc{Parameter}_{\text{Part},\text{Action}}$$

For example, the time to wake up the microprocessor is denoted:

$$t_{\mu\text{P},\textsc{Wake}}$$

A full listing of symbols used can be found below:

Parameter	Symbol
Energy	E
Power	P
Time	t

Part	Symbol
Microprocessor	μP
Sensor (ith of n)	S_i

Action/State	Symbol	Used with:
Waking up	Wake	Microprocessor/Sensor
Active	On	Microprocessor/Sensor
Analog to digital converter sample	Adc	Microprocessor
Execute decision tree	Tree	Microprocessor
Execute designated response	Resp	Microprocessor
Calculate feature	Feat	Microprocessor
Operation Cycle	OP	Microprocessor
Sampling Cycle	Cyc	Alone

Table 1: Summary of nomenclature of variables

Chapter 1

Introduction

1.1 Embedded Wireless Sensing

1.1.1 Background

Embedded wireless sensor nodes and networks are currently being used in a wide array of applications. These include, but are certainly not limited to, detecting degenerative diseases[79], monitoring remote regions[102], and ensuring the safety of housebound elders[52]. 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, limited battery size and continuous sampling of these sensor nodes greatly constrains their lifespan. 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.

1.1.2 Current Status

Size, weight and battery life directly affect user acceptance and marketability of sensors and therefore greatly influence the ability of such technological innovations to improve people's lives and provide societal benefits. In the case of stand-alone sensor systems, the size of the battery effectively determines size of device[1]. As the sensors and associated electronics have been reduced in size, the volume dedicated to powering them (usually via batteries) has remained roughly constant[95]. Sensor systems can both be made smaller and utilized in a larger set of applications if their battery size is reduced by lowering the power consumption.

As well, as sensors nodes increase in functionality (*i.e.* the number and extent of features), they require increasingly frequent use and hence 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 (see section 1.3 for a more detailed discussion) or very simple implementations (see section 3.2). The most common solutions to limited lifespan are to tether the system to wall power[66] or to confine the system to sampling at such a low rate that the lifespan is satisfactory[120]. Obviously, the full potential of wireless and wearable sensors is not being achieved through these systems, as their limited lifespans, sensing capabilities or update rates greatly reduce the utility to the end-user.

For example, long-term medical monitoring is often hindered by its power consumption. Both fixed environmental sensors, which cannot provide a full picture of an active patient's movements, or body-worn sensors, which require large battery packs and/or frequent replacement, are inadequate.

Left solely to the progress of semiconductor technology and battery chemistry, the situation will eventually become untenable and the rate of technology transfer to the consumer will slow or stop, as commercial devices currently use only the simplest forms of power management (see section 3.2). The solution then is to make sensor systems more power-efficient through their design.

[1]For other types of systems, such as PDAs and mobile phones, much of the volume is consumed by the user interface.

1.2 Solution

1.2.1 Overview

This dissertation work improves the capability/lifespan gap in wireless 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 should be central[2]. We concentrated on reducing the energy usage of the sensors (and the associated processing) 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:

- data to process
- data to transmit or store
- data to analyse (particularly for a human expert).

Any gains through this work can be considered independently from the large body of work exploring power savings through improvements to *ad-hoc* networking protocols and processor efficiency. This includes improvements to the software[111], hardware[1], and RF efficiency[69], which are described in more detail in section 3.1.3.

Our goal is to extract the necessary or desired information available in the environment at any given point in time for the smallest outlay of energy. Specifically, 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 determine the system state is collected at any moment. Use of a sensor set which detects the phenomena of interest in multiple fashions and various accuracies further improves the savings by increasing the possible choices for the above process. Overall, the amount of data collected by the system is reduced without affecting the amount of useful information collected.

The form of the 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. Therefore, the work should be applicable to a wide variety of applications.

[2]Fundamentally, these are wireless SENSOR nodes rather than WIRELESS sensor nodes.

1.2.2 Relevant Systems

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[100, 112]. Table 1.1 gives a breakdown of the power usage in three field-tested wireless sensor systems.

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 transmitted directly to a wireless 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 four minutes. The zebras themselves comprise a mobile peer-to-peer network whose goal is to aggregate all 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 - the first level of a hierarchical network which eventually transmitted the data off the island. For each project, power usage is broken down into three categories: the baseline power to run the processor and the wireless link, the power expended in sensing, and the power used in responding to the sensor data. In each application, the percentage of the power used for each of these categories is more important than the total power usage.

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 power for networking, since even though the radio is 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, leading to sensing drawing approximately half the total power

	Baseline		**Sensing**		**Response**		
System	Processor	RF Monitoring	Type	Power	Type	Power	**% Power for Sensing/Response**
Gait Shoe [8, 79]	35 mW	N/A	IMU every 5 ms	65 mW	Tx Data	15 mW	70
ZebraNet [57, 129]	15 mW	13 mW	GPS every 4 min	30 mW	Tx Data	0.2 mW	52
Great Duck Island [71, 116]	118 μW	465 μW	Light, heat, humidity every 5 min	118 μW	Tx Data	14 μW	19

Table 1.1: Power usage breakdown of selected embedded nodes/networks

Device	Base Power Usage	Increase from IMU[a]
Magellan eXplorist 400 GPS unit[70]	275 mW	23%
iPod Shuffle[b] MP3 player[55]	67 mW	97%
Motorola v60t GSM Cell phone[109]	20 mW[c]	325%
Quartz Watch Movement[d][34]	1.7 μW	3.8×10^5%

Table 1.2: Power usage of selected commercial wearable devices with added IMU

[a]From gait shoe as above (65mW)
[b]2nd Generation
[c]Standby power (strong signal)
[d]ETA Trendlife F05.111

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. Finally, since each of these systems is designed for data logging and off-node analysis, their sole response to the collected data is to transmit it wirelessly. The proportion of the total power usage is related to the net amount of data generated, with the gait shoe using a substantial amount of power to offload the data while ZebraNet and GDI both use negligible amounts.

Overall, each system dedicates a significant percentage of its power usage to collecting sensor data and responding to it. Further, it should be noted that the power used to listen to the radio 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, reducing sensing should lead directly to a reduction in both RF transmitter and receiver power usage.

At an application level, there is considerable interest in adding sensors to everyday devices, both for medical[91] and personal activity monitoring[83]. To a first order, the feasibility of doing so would be based on the increase in power usage (and decrease in battery life) caused by including sensing capabilities. Table 1.2 shows the percentage increase in power usage of four common body-worn electronic devices if the IMU from the gait shoe above were to be added to them[3].

It is interesting to note that the power usage of the IMU is significant in each case, even that of a portable GPS unit (which combines high power sensing with an LCD display). As for

[3]Although the power consumed by commercial inertial components has dropped somewhat since the gait shoe was designed, the difference is small enough that it is still a valid point of comparison.

portable music players and cell phones, the IMU's power usage is on the same order as that of the base device. Finally, while it is no surprise that a quartz movement draws negligible power compared to continuous data sampling, the magnitude of the ratio demonstrates that a wearable IMU whose power usage can effectively be ignored (as is the case with wristwatches, which have battery life in years) is not naively achievable. In fact, in all these cases, augmentation of the device with sensors without associated power management would most likely significantly reduce marketability by increasing the frequency of battery replacement or recharging.

1.2.3 Scope of Research

While any sensor system should be able to benefit from the techniques that were developed in this dissertation, it is helpful for the purposes of discussion and evaluation to concentrate on a specific class of systems. This and other related restrictions written into the scope of the problem statement are discussed below. Further limitations to the applicability of the work created by the structure of the proposed solution are discussed in section 2.3. Possible expansions of scope and methods for overcoming the structural restrictions are given in section 8.2 and section 8.1, respectively.

Explicit Limitations

While many application areas can be addressed by this framework, the main constraint applied to this work was to limit our consideration to embedded sensor systems used in human-focused applications - those which collect parameters of human activity either directly (personal/on-body) or remotely (environmental/off-body). These systems would benefit most in terms of usability from reduced power drain as they are required to use batteries because of their mobility or inaccessibility, respectively. Also of importance is the large body of prototype applications in this area which allow for points of comparison.

This achieves two goals. Firstly, the relevant applications are narrowed to what is generally considered the consumer domain (lightweight, cheap and easy to use), corresponding to the scenarios of interest (section 1.3). The scope of sensors and other electronics to consider in the hardware design is reduced in terms of acceptable size and cost, which corresponds roughly to accuracy and update rate for sensors and processing speed for microcontrollers. Secondly, the limitations should render tractable the optimization algorithms, which will be used to generate the real-time embedded software, by narrowing the range of appropriate/acceptable sampling rates as well as the space of useful features. The results of the design and calculations for this class of systems provide guidance to alter the optimization algorithms such that they can be applied to a wider range of systems.

Also, while nodes in collaborative wireless sensor networks could certainly benefit individually from the algorithms developed, this would be a first-order approach. More interesting are the potential benefits to the network as a whole from the nodes informing their neighbours of their current data or sensing state, allowing them to adjust their own sensing to guarantee that the network (rather than the nodes) was collecting the available information in the most efficient fashion. This necessarily requires details of the network size and structure.

We also limit the systems examined to the use of passive sensors - those which measure the environment without affecting it. Active sensors (*e.g.* sonar, radar) - those which control the transmission as well as the reception of a measured signal - are excluded from the initial analysis for two reasons. Firstly, they can use one to two orders of magnitude more power than passive sensors, often making them inappropriate for low-power systems. Secondly, their power management is potentially more complex as both the output power and the sampling rate can be adjusted individually. Section 8.2.3 offers some avenues for incorporating active sensors into this framework.

1.3 Scenarios

While the application space specified is still quite large, there are a few broad categories that comprise the majority of the relevant published work. Within these categories, we consider device scenarios without consideration of lifespan limitations to demonstrate the possible benefits of our approach. We further discuss current implementations in these areas and describe how the ideas developed in this dissertation could improve their functionality.

1.3.1 Medical Wearables

Wearable devices for the collection of medically interesting and important data can be categorized based on the amount of user interaction involved. Those which are triggered by the user - such as a blood glucose monitor - can only be made more efficient through improvement in sensor technology. Those which collect data without direct interaction can be improved though more efficient determination of the user's state, and therefore the value of collecting data, at a given moment.

As an example, we consider a shoe-based wearable system with electronics integrated into the heel and insole. The system captures both the inertial parameters of gait (manner of walking) as well as the pressure distribution under the foot. Changes in gait are surrogate markers for a variety of medically important phenomena, such as developmental maturation, likelihood of falling, and recovery from a stroke. Currently, clinical gait analysis usually is carried out in a confined environment - the patients typically walk less than 10 m per trial - using expensive vision-based motion capture systems. A wearable system, built by our group in conjunction with the Massachusetts General Hospital (MGH) Biomotion lab[79], is smaller and cheaper than these systems and can collect data in any environment. Gait parameters are measured using a full six degree-of-freedom inertial measurement unit (IMU) and an insole with pressure and bend sensors. The full set of sensors is polled at 75 Hz, and the data is wirelessly transmitted to a nearby basestation. The IMU is attached to the back of a patient's shoe via a 4 in wide PVC attachment which protrudes 1.5 in. The attachment weighs approximately 300 g, including a 9 V battery which powers the system for roughly six hours continuously. This system was validated against motion-tracking and reaction force data from the MGH Biomotion lab and produced nominally identical results. However, the lifespan of a single cell is not sufficient for even a single day's continuous usage. This is a product of the continuous collection and wireless transmission of the data[4], which is appropriate for laboratory testing of limited duration where the patient is assumed to always be in motion. For general use, it is necessary to limit power usage while the patient is in one of a number of uninteresting states (standing still, non-ambulatory motion, *etc.*). At

[4]Data storage in a flash memory requires a roughly equal amount of energy[94].

the same time, the continuous sampling, analysis and storage of interesting motions allows for new approaches to long-term medical treatment, both of diagnosed and asymptomatic patients.

In the former case, continuous monitoring can be of benefit in a variety of fashions. Purely passive monitoring can collect data which will help a doctor plot the course of a disease and adjust medication both in type and dosage. Given that the data is being collected, there are also a number of simple parameters which the device can monitor and use to provide real-time feedback to the wearer. For example, Parkinson's disease patients often suffer from a chronically stunted walk and can spontaneously break into a slow shuffling gait. Research indicates this condition improves when patients hear strong rhythmic cues from a metronome[119] or other music[87, 90]. The wearable could easily detect the characteristics of such a gait, and could produce the appropriate beat to reestablish proper stride length and pace. If combined with a portable music device (*e.g.* iPod), it would be possible for these cues to be detectable only by the user, a benefit since most people prefer discrete systems. In the case of motor rehabilitation, the wearable could cue the patient towards correct movements and away from dangerous ones. The system would, in effect, be replacing sensing capabilities which the body had lost and needs to relearn. Finally, even fairly mundane medical problems such as a broken leg could be aided by this system. The normal recovery path involves taking an increasing portion of the normal weight on the damaged limb over time. While this is fairly difficult for the patient to judge accurately, it is trivial for the wearable, which could also keep track of progress and set the timeline accordingly. In this case, the system would be providing sensing capabilities which the body never had.

An example of a mature medical wearable is the cardiac defibrillator[26]. The device itself consists of a processor, lead electronics, capacitors and a battery (which occupies most of the volume), and is implanted subcutaneously with leads connected to the heart. The processor analyses the cardiac rhythm for tachycardia (fast heartbeat) or fibrillation (irregular heartbeat). In the case of tachycardia, small pacing shocks are administered to attempt to restore a normal rhythm. If this fails, or if the heart goes into fibrillation, a much larger shock is applied. The device also records all tachycardic episodes for later analysis. Despite several small pacing shocks being administered daily, the device has a lifespan of approximately four years. Of note is the device's tiered output. While administering the shocks to correct a tachycardia, it is also charging the main capacitors in case of fibrillation. If a normal rhythm is not restored, this charge is immediately available; if not, it is simply dumped. This assures readiness while minimizing the number of painful shocks administered. Such a tiered structure will be used in the analysis techniques of our proposed system.

The more interesting group of patients, in the long run, are those who are currently healthy (or at least asymptomatic). Any device used with such patients would have to be subtler, cheaper and easier to use than those used with the patients mentioned above. The benefits could be just as great, but for a much larger portion of the population. The device could learn the patient's normal baseline over time, providing a valuable reference point which is normally not available to doctors, who tend to only see patients after they become ill. This baseline would allow for quicker diagnosis of conditions which express themselves through simple externally measured parameters such as motion and gait, since the comparison to be made would not be with the normal range for the population at large but the much narrower range of the patient alone. Also, knowledge of the patient's normal state could help avoid false positives as well, allowing a doctor to differentiate between, for example, an abnormal gait caused by mundane influences (such as frequent caffeine consumption) rather than pathological ones.

Long-term tracking of asymptomatic individuals was examined in the doctoral work of Brian Clarkson[23]. The goal of this work was to show that reoccurring events in daily behaviour could be recognized and predicted based on body worn sensors. The specific sensors chosen were: forward and rear facing video (32x24 pixels, 10 Hz), audio (16 bits, 16 kHz) and 3 gyroscopes for orientation (8 bits, 60 Hz). A Pentium III processor and 10GB harddrive were provided to process and store the data (5GB a day), and the system was powered by four large lithium batteries (approximately the same volume as the rest of the components combined) which lasted for 10 hours. In postprocessing, the system recognized 19 different scenes with an overall accuracy of 97%. Activity prediction based on transition probabilities between scenes gave promising results for long-term activity analysis. Increases in the length of scenes such as walking to work could indicate normal aging (slow decline) or illness (rapid decline). Reduction in the number of scenes seen would suggest risk of depression. However, as with the wearable gait laboratory, the prototype implementation presents a number of problems, key among them being device size and lifespan. More efficient data collection is a possible solution as data is collected continuously regardless of state. For example, the addition of more inertial sensors would allow for motion to trigger the camera, reducing both power and storage use for a minimal increase in volume. Such a technique is used in the wearable SenseCam system[43], which uses inertial data to select when to take pictures and to ensure the quality of the pictures by waiting for the user to be somewhat stationary first.

1.3.2 Remote Monitoring

The application space of remote monitoring encompasses those tasks where data needs to be collected about ambient phenomena or local fauna in an environment where the sensors themselves cannot be easily accessed. The data itself is either cached or transmitted wirelessly. This limitation can take the form of distant or hostile environments, where the ability to change batteries and recover data is limited, but also applies to consumer products which perform low priority tasks and therefore must not require large amounts of user attention or maintenance. While the monitoring in these scenarios is not of human activity, the concepts and concerns established in the last section still hold.

In the case of applications for hostile environments, we consider the monitoring and recording of the presence and activity of an endangered species. Wireless sensor systems are well-suited to this problem since they can operate unattended for long periods of time. Further, they often have a small enough profile to be a minimal intrusion. The system would likely collect environmental data (temperature, humidity, *etc.*) at fixed intervals, as well as using heat and light sensors to detect the animal of interest. A small cellphone-style camera might also be used. This data would help estimate the population size, correlate their activity levels with changing weather conditions, and otherwise capture data solely available in the natural habitat (*i.e.* without humans). Many other similar scenarios exist, with the literature most often centring around security and military applications and the tracking of human or vehicular movement.

While there are a number of examples of remote monitoring, a particularly relevant one is the work done on battlefield and border monitoring by a group at the University of California at Berkeley and the Ohio State University[29]. Their work examines the problem of detecting and identifying civilians, soldiers and vehicles travelling through a dense sensor grid in remote terrain. The nodes themselves are 3.5 in x 3.5 in x 2.5 in, are placed roughly 30 m apart and act independently. The area of interest is monitored for activity using a passive infrared detector, which wakes a microphone and magnetic sensor upon trigger. These additional sensors are used to identify the source of the trigger. The project aims for 1000 hours of life on two AA cells, though this goal was not achieved. There are a number of reasons for this. Key among them is that the number of false alarms was far greater than predicted, with false triggers mostly caused by moving flora. This illustrates the danger of designing systems without the use of real-world sample data streams. Also, the system draws more power than necessary since it turns on all sensors after a trigger, while the acoustic sensors alone can be used to distinguish between humans and vehicles and draws far less power than the magnetic sensor. It is further noted that the fragility of this system (in this case the high false alarm probability) is a product of the *ad-hoc* fashion in which the classification algorithms were designed (in this case the setting of the thresholds[28]), while a more analytic approach would likely have both identified the flaw sooner and made it easier to correct. Our solution (chapter 2) presents such an approach.

In the case of low-priority applications, we consider a system for monitoring the long-term stresses on building members and civil infrastructure. Again, temperature and humidity would be of interest, as would pressure and bend sensors and inertial data from accelerometers. Size is again the key parameter, not only to avoid alteration or stress to the beams, but also to allow the sensors to be easily retrofitted to the members rather than having to be implanted at time of construction. The data recovered would prove useful for evaluating and improving structural designs using the force distributions measured under a variety of circumstances. Also, in rare cases, the data could give advance warning of possible damage which could be catastrophic if unchecked.

Building infrastructure monitoring using wireless sensor nodes is still nascent, leading to a lack of useful references. Currently, the two main lines of research are testing the use of established hardware (*e.g.* Motes[9]) which shows that it cannot handle the peak stresses seen[44] and the building of robust nodes from scratch which have not yet been deployed for testing[20]. The state of the research related to civil infrastructure is similar, with deployed systems able to measure conditions but not predict integrity[128] and more complicated systems still in the design stage[17].

The link between these cases is that in both we are most interested in fairly rare data. It is fundamental that the system be able to capture this data when it presents itself, which requires vigilance (*i.e.* at least minimal sampling at frequent intervals). These applications tend to be enormously power-limited, though it is possible to increase the lifespan of such devices by being more judicious about the data sampled. Rather than sampling a fixed amount from all sensors at fixed time periods, it is possible to sample small amounts of data from more appropriate sensors (state determination) and then determine whether to sample more (state response). Therefore, it should be possible to sample a large amount of data at a few time points rather than smaller amounts of data far more frequently. This would have the effect of not only reducing the power usage of the device, but also increasing the relative quality of the data as well.

1.3.3 Distributed Tracking

The final application space we will consider is that of distributed tracking. While this area bears some resemblance to remote monitoring in that it uses nodes to capture data about local inhabitants and conditions, it fundamentally differs in that the nodes interact with each other to complete a single sensing task, rather than collecting data in isolation. Beyond that, the assumptions remain the same (inaccessible environment, limited maintenance ability, *etc.*). While we are not examining distributed applications in this work[5], it is still worthwhile to quickly reference a few projects in the field. Their sensor hardware designs can still be instructive, even if the software and analysis are not directly applicable.

[5]However, see section 8.2.1 for possible extensions.

Consider the case of a network of simple sensor nodes which monitors the movement and activities of the occupants of an apartment. Such an application is currently being proposed as a solution to the impending mismatch between the number of people requiring some form of home care and monitoring and the number of medical professionals able to provide it. Networks of sensors - some video cameras watching portions of the house, others simple switches attached to drawers and so on - would feed their data into a central processor for analysis. In combination, they could provide a reasonable picture of what is taking place in the home at any given time. Using learning algorithms and pattern matching, it is possible to determine patterns which represent common daily activities. It is then possible to determine if those activities have been skipped (*e.g.* taking medicine) or forgotten while in progress (*e.g.* making tea). Except in the case of accidents or emergencies, a central processor manages the problem locally. Since this application is designed to preserve scarce resources, it is important that the nodes be easy to install (most likely a simple retrofit) and require minimal upkeep (*e.g.* battery changes).

Such a system is being tested by the MIT Changing Places/House_n research consortium (Georgia Tech[58] and Intel[50] also have similar initiatives). Within their PlaceLab apartment[54], two different styles of sensors are used to track activities. As above, wired cameras are used for detailed overhead shots. In addition, simple sensors are deployed to measure data at key points in the environment. These sensors are designed to be low-cost, -power and -upkeep data collectors. Each is a circuit board, approximately 1 in square, containing a microcontroller, real-time clock and an input for a binary sensor - either a reed relay switch or a piezoelectric flag[35]. These sensors were attached to controllers, such as light switches or stove knobs, and containers, such as drawers and cabinets. Each time the sensor is tripped, the processor is awakened from a low-power sleep state, writes the current time to memory, and returns to a sleep state. Assuming ten triggers a day, these sensors will last for one year on a CR2430 coin cell, infrequent enough to allow for simple upkeep by a service company. Along with user annotations of example activities, these two forms of sensing are combined to determine the current activity of the home's occupant. The value of sensing specific parameters of interest (*i.e.* the opening of a cabinet) directly is demonstrated by the quality of activity recognition achieved (up to 89% compared to a baseline of 30%) based solely on completely innocuous sensors[81].

1.3.4 Notable Thresholds

In the above scenarios, the potential gains from this dissertation are two-fold. In general, any decrease in power consumption increases the system lifespan, with the concomitant benefits detailed. Further, a decrease in battery use has the added benefit of reducing the amount of chemical waste going into the environment. These benefits, as a goal in and of themselves, are of value societally, even if they are not notable for the individual user.

However, there are two key thresholds in the area of power consumption. The first, which this work should help surpass, is that of commercial viability. Consider the medical wearable device discussed above. An early hand-scripted implementation of the techniques to be developed in this dissertation (detailed in chapter 2) increased the life of the wearable gait laboratory described from approximately six hours to just short of six days[7]. This would allow the system to provide real-time feedback to the user as part of everyday life, allowing for corrections which could help avoid a number of different injuries or aid in rehabilitation[79]. In general, systems which are currently confined to laboratories could be used in unconstrained environments, increasing both their potential applications and users.

The second threshold, that of infinite life, is somewhat further off. However, advances in parasitic power for wearable systems[113] combined with the high-level algorithmic power savings from this work and the inevitable low-level technological gains should eventually make it a reality. Eliminating the need for a power cell will simplify devices, reduce the amount of waste generated, and allow for their placement in areas which are either sensitive (*e.g.* subcutaneous) or inaccessible (*e.g.* battlefields). Other user benefits include systems which would always be ready for use, which is important for emergency gear, as well as reduced cost, volume and maintenance.

1.4 Contributions

The primary contribution of this dissertation is the construction of energy efficient sensor systems through the use of tiered activation levels cued to evolving sensor stimuli. This work has resulted in a framework for automatically creating state determination algorithms specifically tailored to increasing energy efficiency through the reduction of sensor usage. Previous systems have most often used as many sensors as feasible at the maximum possible sampling rate without regard to power and have employed at best only binary hand-scripted techniques to increase efficiency. The consideration of energy cost as part of the system design at an integral level - and the ability to systematically trade off energy and system accuracy - is not only novel, but imperative for the progress of these technologies and applications.

The ability for others to continue this and related work is key to the continued development of low-power and power-aware sensor system. The modular prototyping system used to construct these systems will help other researchers working towards common ends to implement and test their design more rapidly. Similarly, techniques for quickly instantiating tree-based state determination schemes into hardware shortens the design cycle of these systems and reduces a barrier to exploration of different state determination algorithms. Both can therefore be considered important contributions to future progress in the field.

Also of importance is the design and simulation of a wearable medical sensor system using this framework. A large number of applications in this area (as described in section 1.3.1) are positioned to be of great importance and value to the general public over the next decade.

Chapter 2

Form of Solution

2.1 Overview

The main goal of this work is the reduction of energy usage in embedded sensor systems through the creation and demonstration of new tools and algorithms for their design and construction. We will first consider the general form of our solution and the details thereof. The problem space was limited to the sensing of human activity to reduce the scope of possible solutions as well as to allow the incorporation of constraints arising from a fundamental knowledge of human physiology. The design of the device hardware, the software and its instantiation are then discussed. The section concludes with a short discussion of some of the limitations imposed by this solution. We note again that this solution centres around sensor usage, is independent of any power reduction techniques based on reducing RF transmissions or processor usage and can be used in conjunction with them.

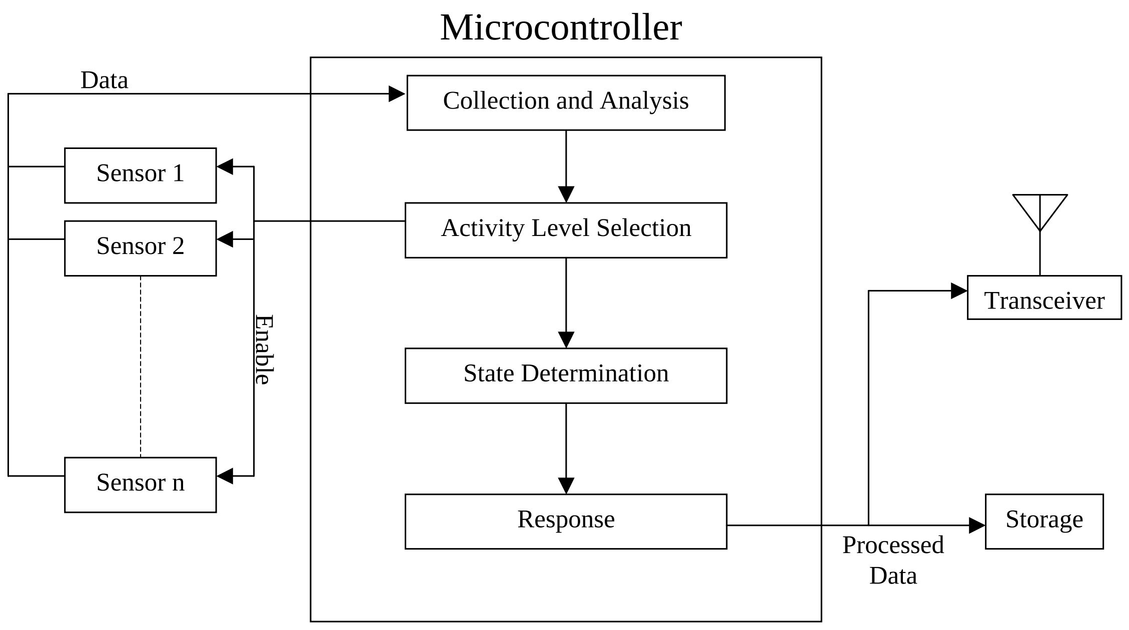


Figure 2-1: Flowchart of proposed hardware solution

The general design for our system is shown in figure 2-1. This system is centred around the concept of what we term “groggy”, or tiered, 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 and their sampling rate, together with algorithms to describe the level transitions. Execution is straight-forward. From a given initial state at power up, the system will begin to collect and analyse the sampled data and will switch activity levels accordingly. At each time step, the current level will specify how to use as little energy as possible to collect the data to determine the current state and whether a level transition is necessary. Specific responses, such as data capture and storage, can be associated with the individual states and are executed next. Once this is completed, the system powers down until the next time step.

2.2 Design Roadmap

Given a desired application, the design process proceeds as follows. First, hardware for the individual application is configured. In the initial deployment for testing and sample data collection, it is assumed that the system will include any sensors that could possibly be of value for state determination. 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 current system state. The sensor set used by the classifier allows the final, possibly pared down, form of the hardware to be built and the data collection to be implemented on it. Each of these tasks are briefly described and commented on in the following subsections. More detailed discussions can be found in the proceeding chapters.

2.2.1 Hardware

As a reasonable starting point for designing a low-power system, the hardware should be as efficient as possible. This generally entails choosing parts with the desired performance characteristics but no greater. Low-power versions of most parts are available at marginally higher cost than conventional versions. Also, a shutdown pin adds extra flexibility in tailoring power usage in real-time and can be quite valuable. If not available on-chip, this functionality can be achieved with a separate CMOS switch.

There are a few specific characteristics to look for when choosing the two most important classes of components for this project: the sensors and the processor. In the case of sensing, parts with analog outputs are almost always superior in power usage to those with digital outputs. Most sensors are fundamentally analog, meaning that a digital output requires extra internal signal processing. Further, since a 12-bit analog to digital converter (ADC) uses roughly the same energy per sample as five processor operations[118], it is more efficient to collect an analog sample in all but the most trivial cases (*i.e.* 1 bit sensors). The other key parameter is sensor wake up time. Since much of the power savings of the framework 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 of how quickly a sensor can be cycled while still offering power savings over continuous activation. For the processor, we consider two main features. The first is the available sleep states. Our design requires an idle state, where the processor can be awoken by either a timer or a change on an external pin, which draws as little power as possible. The time to exit this state should also be as small as possible. The second consideration is the energy drawn per operation. This is notably different than the operating power of the whole device and the calculation of this value should include not only the energy of execution but also the energy necessary to wake up and put the device to sleep.

A 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 counterintuitive that we can make a system more power-efficient by adding complexity (and/or redundancy), the key is that the system has been given a new, lower energy source of information.

The prototype hardware for this work will be implemented using a modular sensor platform we have designed. This platform is based around a series of circuit boards (or panes), each of which instantiates a specific sensing modality - *e.g.* inertial sensing, tactile sensing or ambient sensing. These boards can be arbitrarily combined and recombined, allowing for rapid prototyping and testing of proposed sensor combinations. The individual boards have been built along the lines listed above, with low power amplifiers and analog outputs from the sensors. Further, individual boards generally have redundant sensor sets - *e.g.* the inertial board uses both passive tilt switches and accelerometers to measure motion. For a given application, the designer can use this platform to quickly put together a sensor node with which to collect training data.

2.2.2 Data Collection and Markup

Prior to the classifier construction process, sample data streams must be collected. These streams should be long enough that they contain a suitable number of examples (dependant on the application) of each of the interesting states (those to which the system should respond), as well as enough data from other times to allow the training algorithms to differentiate amongst them.

It is assumed that the application designer will hand annotate the data streams, labelling the areas containing different types of interesting data. We designate each type separately as A,B and so on. The remaining area, where the data is assumed to not be of interest, is labelled X. In the medical wearable example (section 1.3.1), walking could be designated as the A state and shuffling as the B state. The X state would then comprise everything else (stand still, shifting weight, climbing stairs, *etc.*). In general, it is assumed that the interesting examples will be fairly homogenous, while there may be large variations within the uninteresting data. If the designer wished to differentiate between different forms of walking (fast, slow, pathological), they would each need to be assigned to a separate category. However, this would only be necessary if they required different real-time responses (output, data collection, *etc.*).

We have chosen a supervised training approach based on the assumption of fairly constrained applications and clarity of designer intent. Annotations are added to the timestamped data stream by synchronizing with a separate record. This can be done via active schemes, such as a diary or user annotation switch on the hardware, or via passive means, such as a video recording. The former is preferred in most cases as it allows longer sample streams to be collected without regard to location, but passive means are beneficial if the level of user attention to the task is a concern.

2.2.3 Classifier Construction

Given the sample data stream and the user annotated states, the algorithms to determine whether of not the system is in an interesting state can be constructed. We intend to use classification trees for this portion of the analysis[1]. There are three key factors to consider in this form of analysis: the features to be extracted from the data stream, the addition of energy use to the classifier construction algorithm, and the pruning technique. Note that for the application designer, this portion of the process is completely automated.

While the relevant features for any particular sensor will tend to vary, there are a few easily calculated functions which tend to work well across a wide spectrum of data sources. These include mean, variance, minima and maxima. One salient feature of each of these functions is that they can be computed efficiently on a point-by-point basis. Each function is calculated over a fixed window, the length of which will vary with application and will be based on the data markup.

A classification tree is built by recursively dividing the labelled examples into increasingly smaller sets until only interesting or uninteresting examples remain. The splits themselves are simple thresholds on the value of one of the above features, with the chosen split being the one which maximizes a specific criteria (*e.g.* entropy of the data sets). In this case, we are not solely concerned with finding the rules which separate the examples most accurately, but also in doing so for the least energy. Therefore, we will weight the chosen splitting criterion by the energy necessary to obtain the feature used. In most cases, this will be the energy necessary to collect the relevant sensor data as the energy used to calculate the features is negligible. This has the practical effect that once any sensor feature has been used in the tree, all other features based on that sensor's data are, to a first order, free.

The final step to consider is the pruning of the tree both to reduce its complexity and to mitigate the risk of overtraining. This is a fairly straight-forward procedure. Any pruning action will result in lower energy usage and increased response speed in state determination, but also risks lowering the accuracy. This allows for accuracy, power and response speed to be traded off by the designer through simple adjustments in the pruning parameters. At least some pruning is likely to be necessary in most cases, since the lower branches on the tree tend to expend considerable energy to distinguish between a very small percentage of the examples.

[1]This algorithm choice is discussed and justified in chapter 5.

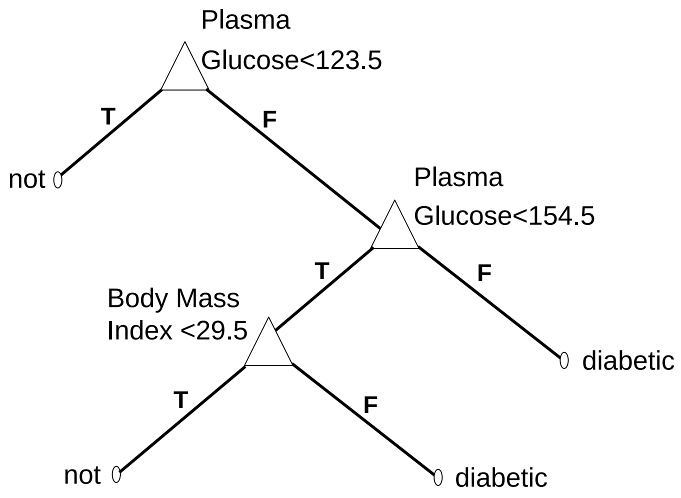


Figure 2-2: Sample decision tree

To make this concept a little more concrete, figure 2-2 shows a sample decision tree for evaluating whether a patient is diabetic. Values of plasma glucose below a certain threshold suggest that the patient is not diabetic, while those above another threshold suggest that they are. For patients with middling values, the body mass index is used to differentiate.

2.2.4 Solution Embedding

The real-time operation of the hardware for this design is fairly straight-forward and is given by the current activity level[2]. A reloading timer is set for the current update rate and the processor is put to sleep. The processor is awakened by the timer overflowing and turns on the desired sensors. As each sensor completes its wake up cycle, it is sampled and then turned off. Once all the sensors have been sampled, the data analysis is done and the state is determined. In the interesting states, the system responds as chosen by the designer, *e.g.* by making a notation of the occurrence, collecting data, or cuing the user. Actions can also be taken in the uninteresting states, though these responses would have to be identical for all possibilities. Once these actions are complete, the processor returns to sleep mode.

There are, as always, a few special cases. At the low data rate extreme, if the sole goal of an activity level is to wait for a transition from a binary (or thresholded) sensor, the processor

[2]The activity level, as described in section 2.1, is the last non-leaf node encountered in the decision tree.

Operation	Example Use	Output Size	Energy Use
Wireless transmission	Raw data transfer	N/A	154 μJ
Fast Fourier Transform	Find spectrum	64 bytes	1.15 μJ
Windowed variance	Feature extraction	1 byte	291 nJ
N-tap feedback network	Generalized filtering	64 bytes	190 nJ per tap

Table 2.1: Comparison of energy cost of common data operations on a 64 byte block

can simply be instructed to sleep until it sees a change on the appropriate pin. Further, this change could be the output of a passive sensor for which the power cost is nearly nil. At the high data rate extreme, it may not be possible to put the individual sensors to sleep between samples, as their wake up time could be greater than the sampling period. It is also possible that the processor itself will not have time to go into a sleep state because of the time necessary to sample and process the data. At this point, hardware redesign should be considered, either to chose more appropriate sensors, shorten the sensor wake up time or the increase the processor speed.

It is straight-forward to algorithmically transform the decision tree found above into microcode. From there, given the structure that we have imposed on sampling rate and feature size, we can automatically generate the transitions between the various levels and the process of activating and deactivating the sensors as necessary. This will significantly simplify the programming and therefore greatly reduce the implementation time.

Once the data has been collected, one response to consider is *in-situ* analysis of data. Most sensor systems either transmit or cache all data without regard to expository value. Simple data processing and feature extraction could reduce this volume considerably for a small power cost. In fact, it can possibly result in a net power savings if data is compressed[5] or discarded altogether. Table 2.1 gives the energy used by the Texas Instruments MSP430F1610 to perform a number of different operations on a 64 byte window, compared to the cost of transmitting that data outright. Due to the variety of applications and desired responses, any use of such algorithms beyond the state determination needed for this framework is best left to the discretion of the individual application designers.

2.3 Structural Limitations

We note a few limitations on the applicability of this work which are imposed by the specific form of the solution presented above. As opposed to the limitations of scope presented in section 1.2.3, these structural limitations can be overcome by altering the algorithm choices. This is discussed further in section 8.1

The key structural limitation of this work is the requirement for a supervised solution for pattern recognition (in the form of the annotated data stream). This approach was chosen based on the assumption of fairly constrained applications and clarity of designer intent. The main benefit is the ability to ascribe specific meaning to the chosen states (*e.g.* walking), to combine states which might otherwise be separated (*e.g.* fast and slow gaits) and to ignore altogether portions of the data stream which could potentially be considered interesting (*e.g.* skipping). On the other hand, systems where the states of interest are either large in number or ill-defined will likely result in trees which are either too deep or classify poorly, respectively.

The tree-based solution requires a binary decision to be made on a function of a block of sensor data at each non-leaf node. Drawing from the limitation of scope to human activity given above, the choice of features was restricted to a few easily calculated functions which tend to work well across a wide spectrum of data sources. While this reduces the complexity of the tree construction, it could possibly increase tree depth (in the case of a combination of simple functions being substituted for a single complex one) and reduce classification accuracy. It is also admittedly slightly incongruous with the principle of guided design applied to the input data stream. The designer is expected to select and label the states of interest, but at the same time not have any mathematical intuition about their structure.

Also, classification trees are time-invariant, *i.e.* state transition probabilities and parameters are not effected by the time elapsed since the last transition (this is not the case for other techniques, such as hidden Markov models[99]). In most cases, there will be no pattern to the state changes, so it is preferable not to have one artificially imposed, though in certain cases (such as people entering and exiting a sensed environment within a fixed interval) it can be of value.

Finally, it should be noted that any wake-up based system has the potential to miss anomalous events (*i.e.* those with no precursor) - either entirely or during the state determination procedure. While this may be a problem in general, it should not be in this case given that we will concentrate on human-centric sensing (see section 1.2.3). Since most activities in this domain take place on the order of seconds (at minimum) and state determination (in our examples) requires roughly a second, the chance of missing an event is minimal[3].

[3]This problem is examined numerically in section 7.4.

Chapter 3

Related Works

We now describe a number of important publications which either implement similar concepts in other fields or which provide background techniques for our work. First, academic publications are discussed - both those directly on point (theoretically and practically) and those from other domains which are peripherally relevant in one fashion or another. Second, we comment on several consumer devices which implement first-order versions of some of the algorithms we have constructed. Works related to various implementation issues will be discussed in the relevant chapters.

3.1 Academic Work

3.1.1 Design Principles

Estrin's[33] survey paper on pervasive networks highlights many of the techniques, both hardware and software, explored in our work. It defines variability, scale and autonomy as the key design axes for wireless networks and nodes. The need to maintain vigilance for interesting data while consuming almost no power is highlighted, and the ability to rapidly alter sensing based on incoming data is cited as the best way to cope with *variability.* The *scale* of a specific application determines the size, sensors and sampling rate needed and are left to the designer in our work. Finally, the need for *autonomy* to facilitate long-term deployment leads to the use of multiple sensors per modality and the desire for a large number of activity levels. Note that Estrin presented only theoretical concepts, with no implementation details given or cited.

Srivastava has written a number of papers [101, 112] which examine the power usage of embedded nodes under a variety of conditions (data collection, wireless transmission, sleep, *etc.*) and comments on the selection of modes for processors and RF transceivers that are most efficient for their particular task. For short-range transceivers, the power drawn by the electronics is far greater than the radiated power. Therefore, data should be transmitted as quickly as possible, with the transceiver shut down between bursts. For the processors, not only should the capabilities mirror the task, but the possibility of never shutting down the processor should also be considered. Using dynamic voltage scaling[92], processor speed and energy can be traded off, allowing the necessary calculations to be performed in exactly the time allotted but for less energy. This work expressly sidesteps the issue of sensor power because the wide variation in sensing modalities, signal conditioning and analog-to-digital conversion make a compact general solution (like those above) unlikely. However, these factors can be taken into account in the sensor hardware design to construct circuits to measure an individual parameter as efficiently as possible. Some recent directions in adaptive sampling are described in [100] and a number of them are detailed below.

3.1.2 Implementation Techniques

Table 3.1 lists the properties of 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. Most concentrate on the measurement and collection of data from a single phenomenon. Jain[56], Liu[67] and Rahimi[102] each use a different model of the sensor data to adjust the sampling rate of a sensor on a single node. Jain uses the innovation of a Kalman filter[42] as a measure of the entropy rate of the data stream and adjusts the sampling rate accordingly. Liu models the system as a random walk and varies the sampling rate when subsequent measurements fall outside of the expected range. Finally, Rahimi measures a phenomenon

Reference	**Application Domain**	Node or Network Level?	Data Collection, State Detection or Tracking?	Uses Model of Phenomena?	**Real-time Adaptation**	Sensors Trigger Sensors?	Nodes Trigger Nodes?	Variable Sampling Rate?	Number of Activation Levels	**Published Details**	Instantiation	Generative Algorithms
Yu[127]	▨	Net	State	No	▨	No	No	No	2	▨	No	Yes
Dutta[29]	▨	Node	State	Yes	▨	Yes	No	No	2	▨	Yes	No
Liu[67]	▨	Node	Data	No	▨	No	No	Yes	∞	▨	Yes	Yes
Rahimi[102]	▨	Node	Data	No	▨	No	No	Yes	∞	▨	No	Yes
Jain[56]	▨	Net	Data	Yes	▨	No	No	Yes	∞	▨	No	Yes
He[49]	▨	Net	Track	No	▨	No	Yes	No	2	▨	Yes	No
Zhao[130]	▨	Net	Track	Yes	▨	No	Yes	No	2	▨	No	No
This Project	▨	Node	State	Yes	▨	Yes	No	Yes	∞	▨	Yes	Yes

Table 3.1: Summary of features of related projects

over a fixed area and uses the spatial rate of change of the data to adjust the spacing of the samples. These are purely entropic 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's work, the Kalman filter is a fairly structured (and computationally expensive) model that would not be appropriate for all systems.

He[49] and Zhao[130] consider power savings in tracking networks with the goal of reducing the total power usage of the nodes. He solves 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. The other nodes may be proactive (they awaken unless they are told to sleep) or reactive (they sleep unless told to wake), allowing a trade-off between power and latency. Zhao examines 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 (using a Mahalanobis distance-based heuristic) 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 continue to respond to information in a very binary way. He turns 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'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.

Finally, Yu[127] and Dutta[29] consider power savings in the more general context of state detection. Yu examines 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 are fused at a central node. Dutta examines 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[1]. Both projects reduce power usage for state detection using the technique suggested in this dissertation - 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 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 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.

The other properties listed in table 3.1 also 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 constructs a hierarchy of sensors, and only the data collection applications consider varying the sensor sampling rate necessary. Finally, there is a paucity of published details both regarding instantiations of the devised algorithms and about how to generate an instantiation of 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 example, looking at wearable examples, Bao[4] demonstrates 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[64] both confirms the above result and shows that multiple sensors on a single node can outperform multiple single-sensor nodes for the same application. In terms of model-based systems, Deshpande[24] presents 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

[1]See section 1.3.2 for a more detailed description.

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. Finally, the Low Power Energy Aware Processing (LEAP)[74] sensor node architecture is designed to allow for a wide variety of power-saving techniques. The power to each subsystem (sensing, processing, communication) can be individually controlled. A central management unit employs a number of power-saving techniques, such as combining the wireless receiver with a low-power paging channel (see section 3.2), adjusting the processor frequency and allowing the sensors to be individually activated based on current data. In the current implementation, the details of the control of the sensing subsystem are entirely left to the application designer (as opposed to the communication system, which is almost completely fixed). The best way of relating our work and the LEAP system is that the classifier output of this framework could easily be used as an embedded sensor management module, with the hardware aspects of control and switching entirely taken care of by the other LEAP modules.

3.1.3 Related Disciplines

While research into power-efficient usage of sensors is fairly new, much work has been done on reducing the power usage of other subsystems. These techniques are of interest both because they can be applied atop our techniques, and because they share interesting parallels with our techniques and may inform our designs. Three areas are considered.

RF techniques

Parallels can be drawn between our system and techniques for dynamic optimality in the operation of RF transceivers in embedded sensors nodes. Work by Marsh[73] contrasts two systems - one which collects data at a high rate but only communicates with a basestation in the case of a specific event and one which collects data at a lower rate but transmits it all. Preliminary testing shows both a notable savings and an increase in event detection accuracy in the former case. Also, Chien[21] examines the benefits of error correcting codes in wireless transmission. Depending on the allowable error rate at a given time, differing levels of encoding (including none) will result in the optimal energy per useful bit.

Many sensor networks, such as the Great Duck Island work[71], use rotating cluster heads - an example of dynamical power optimality. In such schemes, a single node acts as a message aggregator for a number of other identical nodes. This node will use far more power than the others and therefore from time to time this role is reassigned, either randomly or algorithmically, to another node. A similar example in *ad-hoc* networks is the use of randomized routing[31]. While the greedy solution calls for routing through as many (useful) nodes as possible, this would lead to spatially central nodes forwarding substantially more packets than those along the edges. Therefore, these nodes are randomly bypassed to preserve their energy, even though this requires more energy from the transmitting node. While both of these techniques attempt to maximize network life, their approach merely redistributes energy use among the nodes rather than increasing the life of each individual node.

Sleep states and transition policies

In appropriately designed hardware systems, great benefits can be realized simply by shutting down components which are not currently in use. The issues surrounding the shutdown of the RF transceiver are given above. In the case of the sensors, the power management need only take into account wake-up time when turning off unused devices (short wake-up times are key to efficient power-cycling as discussed in section 7.2.3). The case of the processor is more complex because of the variety of different sleep states, each shutting down different peripherals. The most straight-forward approach is to examine the time necessary to enter and leave each sleep state, and to invoke the state with the lowest energy usage that still allows the processor to wake up before the next event. Detailed calculations for this scenario are presented in [110]. If events are asynchronous, it is necessary to predict when the next one is likely to occur. The work in [92] details a number of algorithms which can be used for this purpose.

Low-power electronics

In the long term, the feature extraction algorithms suggested and constructed in this dissertation can be transformed and embedded in hardware to further reduce their energy usage. The most interesting work in this area is being done by the Chandrakasan group at MIT. One software technique proposed is the transformation of algorithms such that their accuracy is monotonically increasing with the number of data points considered, allowing for simple energy-accuracy trade-offs[111]. For example, in finite impulse response filtering, this involves reordering the filter coefficients by magnitude. Note that the accuracy of these techniques can be chosen at runtime based on the needs of the current system state. In the hardware domain, their work has concentrated on techniques that break down a functional block into a number of equivalent smaller systems the most suitable of which can be chosen on-the-fly[10]. For the simple case of a 16 by 16 multiplier, the function would be implemented with a number of multiplication blocks ($1x1, 2x2, \ldots, 16x16$) of which the smallest suitable block is used for any given calculation, trading circuit space for energy efficiency. Given these techniques, it should be possible to specifically design and construct hardware to execute the state determination, sensor control and sampling and data management algorithms far more efficiently than with a general purpose processor. Such special-purpose implementations are common in mass-produced consumer devices and therefore represent an important long-term goal.

3.2 Consumer Devices

There are a number of consumer devices which exhibit similar behaviours to those implemented in this dissertation. The cases of cell phones and personal digital assistants (PDAs) are discussed below.

There are two key points to note when comparing the embedded sensor systems considered in this dissertation to mobile phones. The first is that while talk/standby lifetime is clearly important, it is considered secondary to ensuring quality of service (QoS) to the user.

Therefore, most research is directed towards that end. The second is that both the cell tower and mobile handset are (nominally) under the same control, making this situation more akin to an active sensor than a passive one. Regardless, there are features in cell systems which also have the effect of reducing energy usage to match current needs. First, phone transmission power is set at the minimum level necessary to achieve the desired QoS[103]. This is done to avoid a facet of the near-far problem where terminals near the cell station swamp out the signal from more distant ones. This power adjustment is included in the channel assignment message by the station, and thus the decision is not local to the phone. Second, GSM algorithms (*e.g.*) have been designed to model the current signal interference and noise in real-time, allowing for accurate decoding of signals with lower signal to noise ratios than otherwise possible[69]. Again, the goal is to increase QoS, rather than reduce transmission power, and the sampling rate at the receiver is not altered.

Power reduction in cell phones is more commonly achieved through fixed designs. The radio frequency (RF) circuitry is, not surprisingly, the largest power drain for cell phones[115] and there are a wide variety of low-level design techniques which can increase the efficiency thereof[1]. Further, energy usage can be lowered simply by reducing the time which the RF receiver is on. GSM call initiation[2], both from the handset and the station, is done via a low-power narrowband control channel[41]. The handset monitors this channel with a one-eighth duty cycle, requiring far less energy than reception on the main transmission channels. While this system does use a low power sensor to activate a higher power one, it is a trivial two-state example which relies upon time synchronization between the tower and the mobile. The systems discussed in the section 1.3 (*e.g.*) are much richer and therefore could benefit from a greater number of levels which would allow for a broader range of sensor sampling rates and activation.

Many cellular handsets also provide a range of multimedia services and user interfaces. These features cannot be reasonably distinguished from those of a PDA and are therefore considered together. Power management in PDAs is based, to a large extent, on system timers. The screen will be turned off (or dimmed) after a certain period of user inactivity. It is also possible to sleep the processor, wireless transmitter and disk (if present), while

[2]Similar techniques are used in all cellular protocols.

keeping the screen active using a frame buffer[12]. This interaction is built on top of the Advanced Configuration and Power Interface (ACPI) layer, which is also present in modern computers[46]. We note that this behaviour is scripted ahead of time, and is not altered in real time except by direct user control. Further, the response is binary: any peripheral powered down can find itself fully awakened almost immediately by user input, usually at a net energy cost. More recently, PDA designs have used processors that can be placed in a lower power, though still operating, state by varying the input voltage and clock frequency. Known as dynamic voltage scaling[92], this allows calculations to be performed more slowly but for less net energy. It is most useful in applications, such as video decoding, where operations which take differing amounts of time must be done on a fixed schedule. While the system is adaptive, estimating the time necessary for the next calculation based on previous ones has not been very successful - especially in user interface applications[92] - and often the energy savings compared to simply running at full speed and then sleeping for the duration of the cycle are minimal.

In both of these cases, any power management is implemented either through fixed techniques or simple hand-scripted algorithms. Neither system takes advantage of low-power sensors (which are either already present or could be trivially incorporated) that could allow for a more nuanced interpretation of user input and action. Power savings could be drawn from this knowledge through the tailoring of sensor activity levels. The following sections provide details of our design for the automatic creation of application-specific algorithms which can realize these benefits.

Chapter 4

Hardware Design and Implementation[‡]

4.1 Overall Design

4.1.1 Motivation

As discussed in chapter 1, embedded sensor nodes have become a staple for a variety of applications over the last decade. Recent examples from the Responsive Environments Group at MIT Media Lab alone have included wireless systems to capture the expressive movement of a dancer[89], to quantify the movement of a pair of foam rubber buns for experiments in human-computer interaction[6], and to measure and facilitate group interaction in large meetings[61]. Many of these systems are quite similar, sharing portions of their hardware and software infrastructure. More importantly, they share large amounts of low-level design, in the forms of the sensing, processing, wireless transceiving hardware (discussed in depth in [62]) and software written to interface with or control their functionality. However, each system, because of its unique form factor and choice of sensors, needed to be prototyped from scratch, thereby incurring needless effort in design and debugging. To overcome these

[‡]This chapter is an extended version of a previously published paper[8].

problems in general and simplify the rapid prototyping and testing of embedded sensor nodes, we decided to design a modular sensor platform.

We begin by describing the goals and philosophy behind our design. We then discuss the implementation of the hardware and software. Finally, we provide examples of how a variety of designers have used this system in the past. Note that because this framework takes the form of a series of circuit board connected and stacked vertically, it is colloquially referred to as The Stack (with an individual instantiation called simply "a stack").

4.1.2 Goals

Within the framework of this dissertation, this platform serves a dual purpose: to allow application designers to construct their systems as easily as possible and to incorporate aspects of low-power design that are often overlooked. These goals are incorporated within the design:

Encapsulating knowledge: As mentioned above, the greatest benefit from a modular sensor architecture is the ability to encapsulate knowledge (*i.e.* low-level design). A single board (or pane) of a modular system can encapsulate the best practices of a given field, save a large amount of design time, and allow for easy upgrades. Further, code can be associated with various operations on a given pane, encapsulating them as a functional block rather than simply a hardware block. For example, as radio frequency (RF) transceivers are very sensitive to layout, even the smallest changes can be disastrous. A single pane with a high-frequency transceiver and antenna laid out based on current best practices can solve this problem. The same argument applies to the software for data encoding and decoding, which can often be less than transparent. These benefits can also be obtained on a smaller scale when considering component choice. From the myriad of possible parts, a (small) number will be appropriate in terms of bandwidth, power usage and size (*etc.*). Though their selection is not a technically complex task, it is still time-consuming and subject to error - the reduction of both of which are goals of this platform.

Simplifying prototyping: While the form factor and generality of such a platform may not be appropriate for the final design of some systems, they are certainly acceptable in the testing stages. Therefore, rather than proceeding directly to a stage where the whole system is laid out in its final form, this platform makes it possible to quickly lay out a new pane solely for the application at hand, which can then be attached to other available panes to produce a working version of the new system. This prototype, while likely not optimal for final deployment or mass production, will nonetheless collect the relevant data, provide a valuable proof of concept, help detect flaws in the design, and provide a basis for the construction of necessary interface and analysis software. Further, it is also possible to quickly determine which sensors are of benefit in a given application simply by adding the appropriate panes to the system and examining the resulting output data.

4.1.3 Modular Design Philosophy

The key to implementing the above goals - to making a general system instead of a specific one - is to make the platform as modular as possible. Therefore, the choice of sensors and their layout on the individual panes must be undertaken with care so as to construct functional blocks rather than system blocks. Further, no single subcircuit on a particular pane should be requisite for use of the pane (*i.e.* a combined capacitive-proximity/pressure-sensing pane should allow for use of just one of the two modalities). Ideally, individual panes should be combinations of circuitry that in general either cannot or should not be separated (such as a six-axis inertial measurement unit). Modularity also extends to the individual sensors on each board, where power switches and multiplexers should be used to allow each part to be activated separately. This same modularity should apply to any software written for the individual panes. A single master processor pane will contain the basic software for data collection and transmission as well as communication with other panes. Each of those panes should be associated with blocks of code (or a library) that can be included in the main code when the sensor pane is attached to the processor pane. Also, the software for each pane must be designed such that the appropriate code for any given configuration can be easily composed and compiled.

To be able to fully exploit the modular design, it must be as easy as possible to combine and recombine the available panes into different configurations for different applications.

This will require a simple interconnect system between the boards that allows for repeated insertion cycles as well as for as many signal lines as reasonably possible to run between the panes (to increase the number of possible interactions between them). The goal of simplicity in both the board design and interaction suggest that a direct connection scheme is the best approach. While this requires a central nexus, it avoids the need to place a processor on each pane and provides for much faster data transfer than higher-level schemes (such as ethernet or USB). Mechanically, there are two other requirements: that the interconnects be available on the top and bottom of each pane (allowing the panes to be stacked in any order), and that they provide enough structural strength such that a stack of panes connected together cannot accidentally disconnect, especially in wearable applications where high levels of mechanical stress can be expected.

Finally, for the platform to be most useful, it must be possible for future users to extend it in a variety of ways. Mechanically, this requires that the footprint and height of the individual panes be such that new circuits can easily satisfy those constraints. Further, exclusive use of interconnect lines between the individual panes should be avoided. In the case of the software, the main code needs to allow for inclusion of library files (without source code) for ease of integration. Monopolization of limited processor resources can cause conflicts and should be avoided. Also, the core software for the processor should contain as many helper functions (to set up timers, analog to digital converters, *etc.*) as possible to allow those with a limited knowledge of the particular platform to still be able to code efficiently and quickly.

4.2 Hardware Instantiation

4.2.1 Mechanical Structure

The system itself is comprised of boards 1.4 in square and 0.4 in high, which are interconnected electrically by two headers totaling 26 pins (14 for one, 12 for the other) at opposite corners. The connectors are Molex Milli-Grid shrouded headers and mating receptacles, and are rated for 100 insertion cycles (reasonable for prototyping). The other two corners are used for mounting holes that allow for structural reinforcement of the full stack, which is particularly important for wearable applications. An earlier board layout, while 60% smaller, was replaced because of breakage and general wear of the smaller connectors used.

4.2.2 Electrical Interconnects

The electrical interconnects provide for signal, control and power lines to be run between individual panes. The main method of data transfer is a shared multiplexer bus, though data can also be transferred between suitably equipped parts using Serial Peripheral Interface (SPI). Eight lines connected to the microcontroller on the master board (see below) provide for general direct communication and two lines connected to the external interrupt pins provide for time-critical communication. Finally, four pins transfer power between the boards. In total, this gives:

- Data Transfer (12 Total)
    - Multiplexer (8 Total)
        - 3 Address Lines (8 selections)
        - 4 Enable Lines
        - 1 Shared Output Bus
    - 4 Line Master-Slave SPI Bus
- Control (10 Total)
    - 8 General Purpose Lines
    - 2 Interrupt Lines
- Power (4 Total)
    - +12 V, +5 V, +3.3 V and Ground Reference

Power regulation is handled by a separate board, due to the wide variety of different approaches that can be taken and their respective efficiencies and noise characteristics.

4.2.3 Common Component Selection

There are three components which are expected to be used on virtually all panes: operational amplifiers, switches and multiplexers. These are chosen ahead of time to guarantee acceptable performance and lowest power usage. In the case of op amps, there are three main criteria. The first two are related - a shutdown pin to allow for near zero power usage during sleep cycles and a high slew rate to allow the part to wake up quickly when needed. In general, $0.1\,\mathrm{V}/\mu\mathrm{s}$ is sufficient, giving a maximum $30\,\mu\mathrm{s}$ turn on time. The final criterion is the availability of a single part package. While double and quad packages can ease routing and placement, they also lead to difficulty in making the boards modular in terms of control of the individual sensors. Given these restrictions, the Maxim MAX9911 is recommended. It is available in a single SC70 package ($5\,\mathrm{mm}^2$) with shutdown, and has a turn on time of $30\,\mu\mathrm{s}$. Typical current draw is $4\,\mu\mathrm{A}$ with a shutdown draw of 1 nA. The gain bandwidth product of 200 kHz is acceptable for most uses.

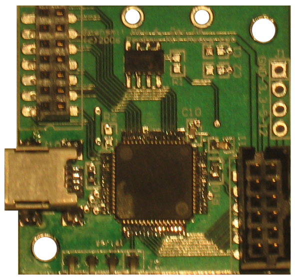
For the switch, low impedance and fast switching are key to controlling power to the sensors. Because of the number of control lines available, a serial interface is also required. Currently, the best part in the domain is the Analog Devices ADG714[1] octal single-pole, single-throw switch with $2.5\,\Omega$ impedance and 20 ns switching. The device is controlled via SPI. The $10\,\mu\mathrm{A}$ typical current draw is somewhat high, since the device runs continously[2]. The prototype ADG791A is far superior with a continuous draw of only 1 nA (and otherwise equivalent specs), but can only pass 3.3 V maximum (even with a 5 V supply[3]). Switching is currently implemented only on the inertial measurement board, as it was the board used for the testing of this framework.

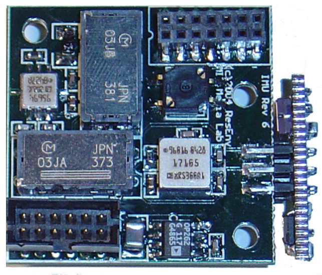
Finally, the multiplexer has similar requirements to the switch, requiring fast switching and low impendence. Control is via the shared addressing bus detailed above. The Analog Devices ADG608 is currently used for the individual panes. It is an 8:1 multiplexer with $30\,\Omega$ impedance and 120 ns switching. The maximum power draw is $0.6\,\mu\mathrm{W}$ when enabled and $0.15\,\mu\mathrm{W}$ otherwise.

[1] The Maxim MAX395 is virtually identical.

[2] Since there is no straight-forward way of activating this part only when needed to control the sensors short of adding another switchable power line.

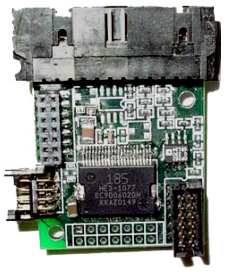
[3] An additional FET pulldown would be needed to accommodate 5 V devices.

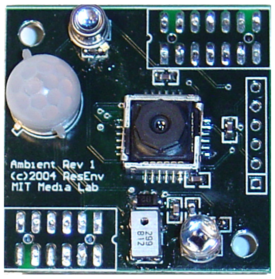

(a) Master



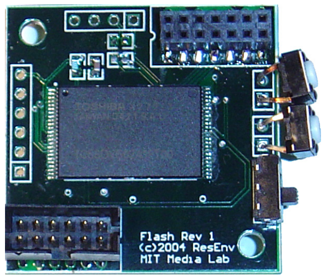
(b) IMU

(c) Sonar Ranging


(d) Tactile



(e) Ambient



(f) Storage

Figure 4-1: Six boards in the modular architecture

4.2.4 Individual Panes

Master: The master board (figure 4-1a) is responsible for the data collection and transmission to a central basestation and is included in every stack. It currently contains a 8 MIPS 16-bit Texas Instruments MSP430F1610 processor with 12-bit ADC as well as an RFM DR3000-1 916 MHz transceiver running at 115.2 kBps. The processor pins are broken out to the interconnects mentioned above, with the general purpose lines and the multiplexer output connected to the analog input pins (which can also act as digital I/O). The processor has two key features. First, it has many low power modes from which it can awaken quickly (*e.g.* 6 μs to awaken from the 8 μA sleep state). Second, the two-cycle hardware multiplier allows the features, such as variance, to be calculated very quickly and cheaply.

The central basestation itself was also built using this master board to manage a simple time division multiple access (TDMA) wireless protocol (described in [79]). While it can technically handle an arbitrary number of stacks, the practical limit is determined by the size of the data packet from each stack and the desired update rate.

Board	Function	Part Number	Accuracy (bits)[a]	Voltage (V)	Power Usage	Wake Up Time	Notes
Master	Processor	MSP430F1610	N/A	3.3	12 mW	6 μs	Continuous operation with 8 MHz clock
Master	Transceiver	DR3000-1	N/A	3.3	15 mW	200 μs	200 - 16 byte packets per second
IMU	Accelerometer	ADXL202JE	10	3.3/5	1.9 mW	8 ms	
IMU	Accelerometer	SPSF1000	1	3.3	Nil	Nil	Static tilt switch
IMU	Gyroscope	ADXRS300	8	5	30 mW	35 ms[b]	
IMU	Gyroscope	ENC03J	10	3.3/5	10 mW	40 ms[c]	
Ranging	Sonar Transmitter	MSI1005853	N/A	12	60 mW	75 μs	Power usage of whole board with 50 Hz pings
Ranging	Sonar Receiver	V-MA40A5R	10	12	60 mW	75 μs	Power usage of whole board with 50 Hz pings
Tactile	Capacitive Driver	MC33794	?	12	50 mW	5 ms	
Ambient	Light Detection	BPX43	12	3.3	50 μW	15 μs	
Ambient	Light Detection	SFH314	12	3.3	130 μW	8 μs	
Ambient	Heat Detection	442-3	10	5	10 mW	2 s[d]	
Ambient	Acoustic	SP0103	10	3.3	1 mW	5 ms[e]	200 Hz high pass filter
Ambient	Camera	FPDB0	16	5	90 mW	100 ms[f]	
Ambient	Processor	C8051F121	N/A	3.3	150 mW	N/A	100 MHz clock to handle camera data
Storage	NAND Flash	TC58DVG02A1	N/A	3.3	13 mW	25 ns	6 - 512 byte pages per second
Storage	Processor	C8051F206	N/A	3.3	27 mW	N/A	

Table 4.1: Summary of major components of the modular circuit boards.

[a] If not specified, found using $\log_2(\frac{2.5 \times Noise}{Range})$

[b] Very high but necessary for planar layout.

[c] Measured by hand. Time for output to stabilize to reference.

[d] Measured by hand. Time for output to stabilize to reference.

[e] Measured by hand. Time for internal amplifier to stabilize.

[f] First three frames invalid after wakeup (at 30 fps)

IMU: The sensor board shown in figure 4-1b is a six degree-of-freedom inertial measurement unit (IMU). Acceleration is measured via the analog output of two Analog Devices ADXL202[4] accelerometer (±2 g), one of which is attached to the side of the pane to achieve the third axis of sensing. A four-way static tilt switch sensor (ALPS SPSF1000[5]) provides for additional micropower single-bit acceleration measurement. Angular velocity is measured via two Murata ENC03J gyroscopes and a single Analog Devices ADXRS300 gyroscope (all ±300°/ sec). This combination allows for full 6-axis inertial sensing in a nearly flat package.

Sonar Ranging: Distance measurements can be achieved using a matched pair of sonar receiver and transmitter boards. The transmitter board sends a single 40 kHz pulse from an omnidirectional transceiver (MSI 1005853), which is then received by two pickups (Gibson Tech V-MA40A5R) board placed a fixed distance apart on the receiver. A measurement of differential time-of-flight is allowed by synchronizing to the basestation's TDMA messages, hence the two receivers allow both displacement and relative angle to be calculated[79, App.E]. It is pictured as figure 4-1c.

Tactile: A fourth sensor board is shown in figure 4-1d and allows for inputs from a number of different tactile and pressure sensors. It includes inputs for four single-ended force-sensitive resistors (FSRs) via common-collector BJT amplifiers, two back-to-back FSR bend sensors via a differential op-amp pair, and two piezoelectric sensors via common-drain FET amplifiers. This pane also contains the circuitry for a Motorola MC33794[6] 9-channel loading-mode capacitive proximity sensor[30]. These are attached via a header at the top of the board, allowing them to be spatially distributed as desired (such as in sensate gloves or shoe insoles).

Ambient: The ambient sensing board (figure 4-1e) provides a range of methods of detecting audible and visible occupants of the local environment. This includes a narrow cone (10°) Osram BPX43 phototransistor and a wide cone (70°) Osram SFH314 phototransistor to

[4]Though this part is now deprecated in favour of the pin-compatible ADXL203, it is retained because its startup time is 4 ms faster. The preproduction (as of Q1 2007) ADXL323 has comparable specs with half the power draw.

[5]Deprecated with no suitable (*i.e.* compact multiaxis package) replacement.

[6]The MC34940, now available, is more appropriate for this application.

detect reflective objects at different horizontal distances from the board. Dynamic heat sources, often (but not always) humans, are detected with an Eltec 442-3 pyroelectric sensor. Finally, a complete visual of the environment can be acquired using the ALPS FPDB0 VGA camera module with dynamic frame adjustment. Acoustic pickup is provided by a SiSonic SP0103 microphone. An SiLabs C8051F121 is provided for *in-situ* signal processing (*e.g.* of camera data).

Storage: On-board data storage is provided by the board shown in figure 4-1f. Data is stored in a 1 Gbit flash memory chip (Toshiba TC58DVG02A1) and I/O is controlled by a SiLabs C8051F206. Data is input to the board over SPI and is output via an RS232 connection.

The parameters of the major components for each of the boards are listed in table 4.1. Because these boards were designed for different purposes and at different times, there are a few component choices that do not fit with the rubric of low-power design. The most obvious is the processor included with the ambient board for processing of the camera data. Because it is clocked at the limits of its capabilities, but draws far more power than necessary. Either a special purpose camera processor (*e.g.* Freescale MX21) or a chip designed to run at hundreds of megahertz would be more appropriate. Further, the two SiLabs processors (on the storage as well as ambient boards) lack a reasonable low-power mode, with the idle state drawing 75% of full power. Switching to an MSP430 class processor (as on the master board) should yield notable savings. Other improvements, such as active illumination for the ambient board, are discussed in section 8.1.2.

It should be noted that this selection of boards merely represents the specific sensors that were necessary for projects constructed within our research group. New boards can be easily created and source code examples and we provided PCB templates for this purpose. A number of applications designed with this platform (and new boards created for them), both by our group and others at the MIT Media Lab, are discussed in section 4.5.

4.3 Software Instantiation

This architecture was previously used exclusively for wireless data collection and streaming transmission. Running on the master board, the sole purpose of the software was to collect the data from the various sensor boards and transmit them to the basestation. Since a TDMA scheme was used, these actions were taken in response to a prompt from the basestation. The rest of the time, the processor idled (awaiting commands) with the radio in receive mode.

The software itself was constructed in a modular format and contains three main sections: initialization, data collection and data transmission. The addition of a new board to an application can be accounted for in the code by adding an appropriate routine to each of these sections. A header file may also be necessary for local variables. Also, macros and helper routines exist within the main code to aid in common operations. These include data collection tasks, such as setting the channel on the shared multiplexer bus, mutual exclusion on the multiplexer enable lines, and ADC sampling. Data transmission is facilitated by a hard-coded 6 to 8 bit DC balancing scheme (for use with the 12 bit ADC data), as well as by macros for SPI and UART communication. Miscellaneous tasks such as timer start, stop, store and reset and port input/output mode selection are also handled.

For a stand-alone application such as those considered in our work, the changes are fairly minimal. We have removed the data transmission and the associated infrastructure. In its place, code is added to perform the specific tasks desired by the application designer. The initialization and data collection portion of the software are untouched.

4.4 Related Works

Other research projects are currently working towards similar ends and producing similar systems. However, each is attempting to solve a slightly different problem, leading to important differences.

The best known system in this space is the Telos Motes[93] designed at UC Berkeley and produced by moteiv[80]. Each Mote is a 1.25 in by 2.5 in board with attached power source, processor and wireless transmitter. Low power usage is achieved through careful component choice and (mostly) by running on a 1.8 V supply. This main board can be supplemented by a single expansion board via a six and a ten pin header which provide analog inputs, communication (serial and I^2C) and power. No sample expansion boards are provided and the choice of sensors is limited by the supply voltage. This approach eschews modularity for the sake of size and integration, since incorporating another degree of sensing requires redesigning the expansion board or the adding another wireless sensing node. Further, the associated research has concentrated much more on building an *ad-hoc* peer-to-peer network of these boards, rather than the collection of data for on-board or central processing.

The Power Aware Sensing Tracking and Analysis project (PASTA)[107] consists of a number of stackable circuit boards, each 1.75 in by 2.5 in. PASTA is designed for applications an order of magnitude larger than ours. Processing is done via a 400 MHz XScale processor and separate boards provide for multi-channel analog to digital conversion and digital signal processing. They are connected together through a 180 pin header consisting mostly of direct parallel communication buses. The only available sensor board is an acoustic vehicle detection system, though an interface for Mote sensor boards is available. PASTA has an impressive thousand fold difference between full operation and standby mode, aided by the ability to individually control power to each board. However, the minimum power usage is still nearly 1 mW (compared to 8 μW for our system) with the sample tracking application drawing 178 mW on average. While these are impressive results for their domain, PASTA creates bigger modules which require much larger batteries than the applications of interest in this work. Further, the connector design does not appear to have mechanical strength in mind (with connections on only one side of the board and no mounting holes), making wearable applications difficult.

The Small Autonomous Network Devices (SAND)[86] project at Philips is quite similar in goal and design to our system, though it was built with more recently available components. This allows their system to be much smaller, with each circular circuit board only half an inch in diameter. Panes are available matching our major functionality: inertial measurement, data collection, storage and wireless communication. More powerful modalities, such as the ambient board, have not been built and most likely cannot be. The main limitation on this system is the battery, a CR1225, which sets the board size and has a maximum constant discharge rate of 1 mA[104]. Even using two in parallel, most of the wireless user interface applications for which The Stack was designed would be impossible with this system.

Finally, the Tower project[45], also at the MIT Media Lab, is in the same genre, at least in some respects. The Tower features a main processor board to which multiple extensions can be added. Each board is designed towards a single input (*e.g.* light sensors, microphones) or output (*e.g.* LEDs, speakers) functionality. The whole system is programmed and accessed via a real-time command line interpreter running on the main board. This system is designed mainly for exploration and building, rather than for testing and deployment. Therefore, the boards are quite large (about 3 in square) and stacks of boards can grow to be 6 in or taller. Since the system is wired, power usage was not a design concern.

In contrast to the projects described above, our work concentrates on the sensor portion of the design, rather than networking or pedagogical concerns. Further, our system was intended primarily for prototyping devices for real-time collection of sensor data and state detection therefrom. This requires a small low-power system with high modularity with respect to the sensor subsystems.

4.5 Sample Uses

To determine whether this platform meets the goals of encapsulation and ease of prototyping, we consider a number of applications built using this device with only minimal assistance from the author. Two applications - one complete, one prototyped - and their relation to the design goals and philosophy are discussed. Further examples can be found in appendix A. Since the designers discussed all have a technical background but had limited experience with electronics design, they are a good approximation of the 'application designer' referred to throughout this document.

4.5.1 Wearable Gait Laboratory

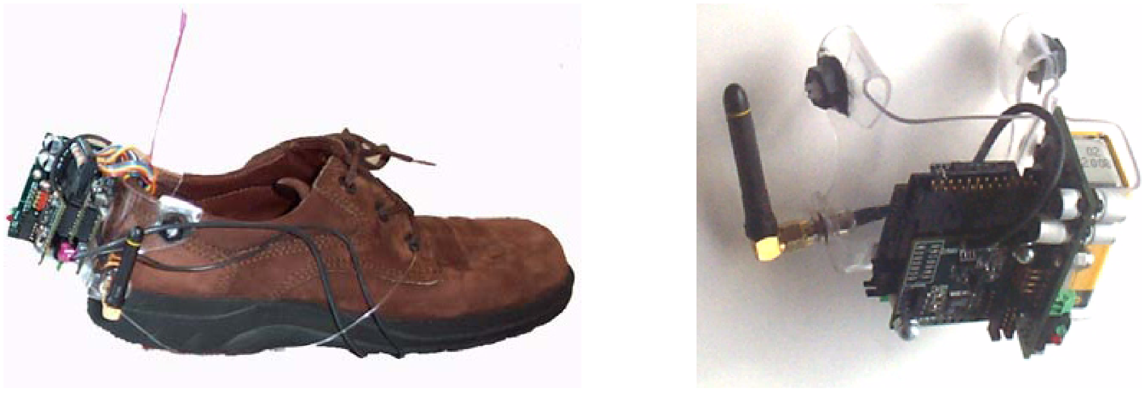


Figure 4-2: Wearable gait analysis platform and sensor attachment

Stacy Morris Bamberg applied this platform to develop a prototype inexpensive wireless wearable system for the analysis of the motion of feet during gait (described in section 1.3.1 and detailed in [79]). The system consisted of the master, IMU, tactile and sonar boards, as well as a power regulation board. The Stack was screwed down to a piece of thermoformed plastic, which was connected to the patient's shoe using plastic screws (figure 4-2). A sensor insole was connected to the tactile board via the header and was placed inside the shoe.

This application took advantage of the extensibility of this architecture in a number of ways. To increase the mechanical strength of the system, a third mounting hole was added to the IMU board. Also, the sonar board was added to the system near the very end of the design revision cycle to increase the accuracy of the foot-to-foot distance measurements (compared to the IMU). In both cases, none of the other boards needed to be altered in any way to accommodate these changes.

4.5.2 Prototyped Systems

Not all applications use the boards from this platform for their final implementation. Often our boards are used simply for prototyping and design, and the circuitry is then redesigned depending on the particular physical constraints of the systems. A recent example is commented on below.

The Huggable[114], designed by Dan Stiehl, is an instrumented responsive plush bear designed to act as a companion animal. It is intended for use in wide variety of settings including those, such as a hospital, where pets are normally not allowed. Along with a number of proximity and pressure sensors (to detect if the bear is being petted), the Huggable uses inertial sensors to determine if it is being held or rocked. This functionality was prototyped using the IMU board prior to being integrated with the rest of the electronics.

The modularity of the architecture was exploited here, allowing motion detection to be quickly tested in a system which had previously been static. More importantly, the encapsulation of expertise in inertial measurement proved invaluable. It was discovered during the testing phase that the tilt switches, which most likely would not have been included on a board designed from scratch, were much more effective at measuring rocking than the accelerometers. Thus, the generality of our design allowed the application designer to locate a sensing solution which would otherwise have been missed.

4.6 Summary

We have developed a compact wireless modular sensor platform, which contains a number of circuit boards (panes). As opposed to similar architectures, this system treats the sensor panes as discrete design objects that have data collection as their primary goal. Six boards have been designed so far: master (processor/transceiver), tactile (pressure, bend, proximity sensing), inertial measurement, ambient (visual and audio), sonar and data storage. These boards encapsulate design knowledge and allow for rapid prototyping of applications. A number of major applications, including a wearable gait laboratory, have been built and user tested. Also, a number of systems were first prototyped using this platform before being implemented in a more compact fashion. Therefore, it is believed that this hardware platform, with minor modifications such as those made to the inertial board, will be more than adequate for use by application designers. In chapter 7, we will discuss a test application for our framework built using this platform.

Chapter 5

Pattern Recognition Design and Implementation

In this chapter, we discuss the core of the design process for creating power-efficient sensor systems. To begin, one (or more) sample data streams containing segments of both the interesting and uninteresting states are collected. These streams are annotated (*i.e.* the segments are labelled with states) by the application designer, features are calculated and examples are extracted. These examples are then used to build a decision tree classifier, with a parameterized power/accuracy trade-off left to the designer. These processes are described in order, and details of design choices are given in the appropriate sections.

Note that there are a number of references in this chapter to the main test scenario examined in this dissertation - a wearable gait monitor. A complete discussion of this application is deferred until chapter 7, though the reader may benefit from referring to it from time to time.

5.1 Sample Data Sets

As a first step, a set of examples must be collected. Each example is a series of values with a given label (in this case, the state) and 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. Finally, features need to be calculated to reduce each complex state to a simple and roughly complete set of examples.

5.1.1 Collection

While data collection is an open-ended process, we propose the following two part procedure to acquire the most relevant data. The first part will be reasonably short and collects a set of data that contains the active (high energy/variance) states, both interesting and uninteresting, that are known to the designer. As this is a supervised learning system, the selection of states is left to the discretion of the application designer. The second part contains a set of long-term background recordings, to provide a baseline for the uninteresting cases and to catch states which were not considered by the designer. All streams are captured at the maximum useful data rate. For human applications, this is pegged at 200 Hz[59] (*i.e.* a Nyquist frequency of 100 Hz). Also, 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 can either be stored locally (using the storage board) or remotely (using the master board to communicate with a basestation).

The purpose of the active data set is to provide a suitable number of examples of the complex states of the system. The meaning of complex varies by application, though in every case the extrema are defined by the interesting states[1] (regardless of metric). A firm definition is not required - for a given case, the designer should have a reasonable idea of those states which are similar enough to the interesting ones that they are likely to require the most effort to distinguish. Rather than wait for these states to occur naturally in the operation of the system, it is most often easier, when possible, to simply capture them directly in a scripted

[1]While uninteresting states may be more complex, there is no need to examine them in any more depth than necessary to differentiate them from the interesting ones.

sequence. This guarantees their presence, quality and labelling. Enough data should be captured to represent the variation within the states. As a rule of thumb, roughly 100-200 examples per state should be adequate, though this value will depend on the complexity of the states and the number of features. Finally, in a scripted sequence, passive means of record keeping, such as video recording, are suitable, since the data collection will likely take place in a confined space and the user's attention will be occupied with performing the tasks at hand.

The active data streams, by themselves, might be sufficient to train the classifier. However, the response of the system to states not present in these streams, either by design or by omission, is unknown. The purpose of the long-term data set is to capture such states. Those not included by design are most often the simpler states which are easiest to separate from those of interest, while those not included due to omission are the states which were either too rare or oblique to be anticipated and included above. This stream should therefore provide a measure of completeness which is lacking from the scripted stream. It can further be used to provide prior probabilities for the classifier. Since the data collection is expected to mimic actual usage as much as possible, it will likely have a time scale on the order of hours or even days, depending on the frequency and variation of background events. Therefore, an active method of record keeping, such as a diary, is most appropriate due to the likely sparseness of states to label and the lack of any limitation on the locale of the collection.

While it is possible to use only long-term data streams as the source of the training data, this tends to be inefficient for a number of reasons. Firstly, the length of the recording necessary to acquire good examples of all complex states may be quite long and their duration quite short. Secondly, the vast majority of the uninteresting data collected will be of little to no value in the classifier training. While having examples of all possibilities is beneficial, this technique is likely to collect many more examples than are useful. Finally, the hand annotation of such a long stream would be burdensome to the designer. Studies by Intille[53] support this data collection scheme.

5.1.2 Annotation

After collecting the data streams, it is the task of the application designer to annotate them. From the records kept, examples are marked on the collected data stream with the appropriate label. We consider the two stream classes defined above.

For the active stream, the designer first converts the collected record into individual times marking the beginning and end of each activity. There will often be small gaps when the recording is turned on and off, and during the transition between states. Given these time points and knowledge of the fixed sampling rate, it is straight-forward to label the data stream with the states. It is best to keep the recorded segments short, as missing data, usually from glitches in the storage or transmission, can skew the timing. While it is possible to resynchronize by visually inspecting the data stream for the transitions[2], this is an inherently subjective technique in most cases.

For the long-term data stream, the form of the labelling will vary with the information recorded. If only active portions of the stream are marked, the rest is considered a single uninteresting (X) block. If all segments are labelled, then the uninteresting blocks are separately designated ($X_1, X_2, \ldots$). In either case, it is assumed that the data stream is long enough that a skew between the record and the data points is inevitable (from missing data, jitter in the microprocessor clock, *etc.*). Therefore, alignment must be done based on cues purposely introduced into the data stream (such as tapping the foot in the wearable gait example) or from the active examples (if they can be clearly differentiated from the rest of the data by inspection). However, it should be noted that exact annotation of the long-term stream is not a necessity. For the background data, as long as the designer errs on the side of including too little data in the marked segments, the risk of mislabelling is low. Further, if the active portions are also included in the first stream, it may be best to simply ignore them altogether rather than possibly include data from an uninteresting state (at the beginning or end) in the example. In both cases, it is up to the application designer to decide if enough examples are available for training the classifier such that a conservative approach is acceptable.

[2]This is usually done by noting longer pauses at the transitions, though it may be possible to track changes in features of the data as well.

There are two further points to consider. The first is transitions between individual states. As suggested above, this data will usually be in an unlabelled gap and it is best to exclude it from the training set altogether[3]. These examples are amongst the most difficult to classify, and will tend to force the classifier to overtrain. Also, the transitions themselves are most often too short to recognize because of the latency associated with the activation of individual sensors (see chapter 6). The second is the labelling of uninteresting states. For complex states, separating the X_i will likely prove beneficial when estimating the average power usage of the system, since the different states will vary in the energy used to classify and in their frequency of occurrence. For the simple states, it is assumed that the same amount of power (*i.e.* the minimum) will be used to classify all of them, and therefore the subdivision is of little value.

5.1.3 Feature Extraction

To train the classifier based on the marked examples, a set of features is extracted from the data. These features should be compact while still representing the variation between the states. We consider the choice of features and the parameters thereof.

Features Used

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 solution will be less than robust. The cause is variation, both from the structure of the time series (and by extension the human activity) and from the noise in the individual data points. To take this variation within single states into account, we will use windowed functions to calculate the features.

We chose to use a set of simple first order functions to calculate the features - specifically, the windowed mean, variance, minimum and maximum. These features have been used

[3]While these situations will, of course, exist during the operation of the embedded classifier, anti-thrashing code (see section 6.2.3) will take care of them.

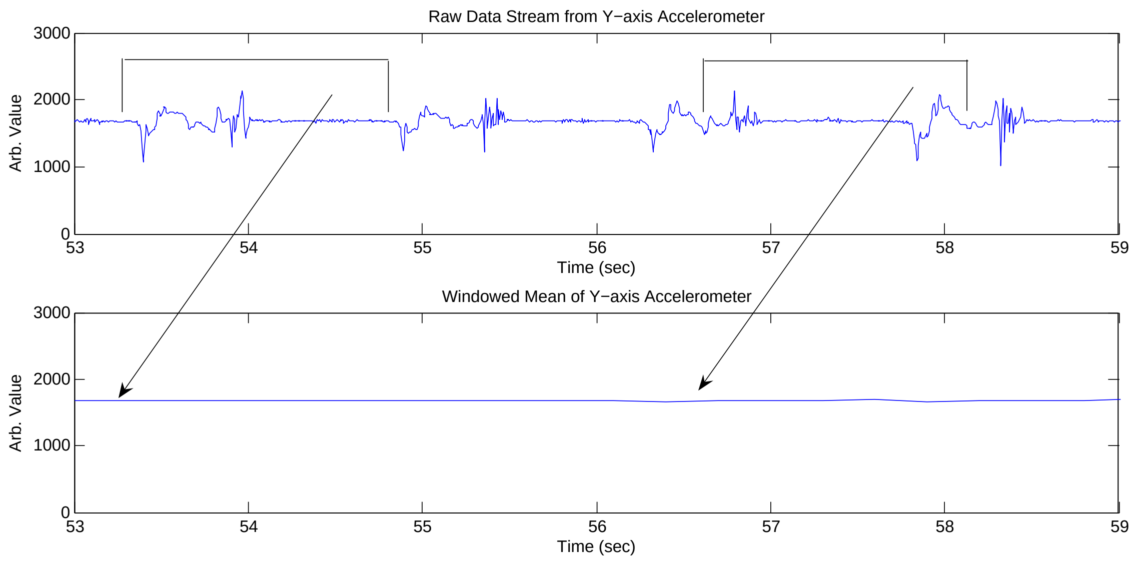


Figure 5-1: Sample data stream and windowed feature

successfully on time series of human motion, with both inertial[2, 64] and video[63] 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 the microcontroller. 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 simplify the classifier by ignoring this portion of the test cost (see section 5.2.3). Figure 5-1 demonstrates the calculation of the windowed mean of a sample data stream (described in section 7.1). The brackets show the extent of the window and the arrow designates the associated point in the feature stream.

We argue that 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 a multiple of the period (or greater than the correlation length), 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.

A few other features bear mention. Zero-crossings and integrals under peaks of the curve have proven quite valuable in previous work in inertial gesture recognition[6]. However, unless the data is zero-mean[4], these functions are far more expensive to calculate ($O(n)$) because they require both baseline (mean) subtraction and absolute value calculations at each time step. Also, many characteristics which are obscured in the time domain can be easily extracted using the chosen functions in the frequency domain. However, the time complexity of converting to the frequency domain (at best $O(n \log n)$ with the fast Fourier transform[16]) limits the utility of this approach.

Parameters

To generate examples using the above features, two more parameters are necessary. The first is the window size, the second is the sampling frequency. Each is considered in turn.

As stated above, for the functions to generate well-behaved features for periodic data, the window size should be a multiple of the period. In practise, this means that the window size is set equal to the period, since the window size is also equivalent to the minimum delay in activating a sensor (as the system must be causal). It should be noted that the period of the data is often not constant, both within and between states, as a result of variations in both the activity and data sampling. As shown in section 5.3.1, this leads to an increase in variance in the feature values. Since the root causes are unavoidable, it is best for the application designer to simply select the period based on a representative example of the interesting state. It is possible to tune this value by varying it over a short range ($\pm 10\%$ of the chosen value) and choosing the one which minimizes the variance of the features.
For non-periodic data, the choice of window size is far easier, as it is simply the value at which the variance of the features reaches a plateau. A plot similar to the one suggested for the period should be adequate to find this value.

[4]In fact, for a single-ended ADC (as used here), zero-mean data would be uniformly zero. However, even the assumption of constant mean (over all states) is unreasonable for most applications.

Given the window size, there is also the matter of the spacing between the generated examples, which can reasonably range anywhere from a single point to an entire window. The goal is to extract as many uncorrelated examples from the data stream as possible[5]. We use a sliding offset of a quarter window length for each example, which has given good results[4, 25]. If the period or correlation length is nearly constant over the stream, it may be necessary to add jitter (by moving the starting point of the window randomly by a few points forward or back) to avoid correlation. This is considered unlikely, however, for the reasons stated above.

Since power use is correlated to sampling frequency[6], examples are generated at a number of different frequencies. The classifier will then choose the lowest power example which has the necessary information (see section 5.2.3). Therefore, the maximum frequency for generating examples is the sampling frequency of the training sets. While any lower frequency is acceptable (as long as its ratio with to maximum is rational), we limit the allowable choices to simplify the programming of the embedded code (see chapter 6). Specifically, to avoid having to power cycle the processor on a complex and irregular schedule (which will vary with activity level) to accommodate mutually prime sampling rates, we only allow downsampling by powers of two. While this may reduce the power efficiency of the system (there will usually exist a frequency between the two available which contains the necessary information for lower cost), the difference is believed to be slight and balanced by the power savings in the processor.

Note that data taken at the lower sampling frequencies will suffer (or possibly benefit) from signal aliasing, since the sensors which make up The Stack are filtered only to remove frequencies above the Nyquist frequency of the phenomena measured. This was a conscious design decision made based on the power cost of varying the cutoff frequency (using a switched capacitor circuit or special purpose chip) and the possibility that data sampled at multiple frequencies will be desired simultaneously.

[5]Correlated examples are actually not a large concern during the training process. However, since the training, pruning and testing sets are drawn (without replacement) from the same collection of examples, correlation of examples between the sets can lead to overfitting during the pruning and testing process.

[6]At least until it is no longer possible to power cycle the sensor quickly enough, at which point the sensor is continuously on (or off) and the power usage is constant.

5.2 Data Classification

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. The choice to use classification trees for this purpose is first explained and defended. The issues surrounding priors and weights in this application are discussed. The algorithms used to construct the tree are then explained, including the splitting criterion, the addition of a weighting based on test cost, and the pruning of the tree. Finally, we present a few simple tests which were performed to confirm that the classifier had the desired properties.

5.2.1 Classifier Selection

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.

Therefore, decision trees with be used in this framework. Decision trees structure classification in the form of a series of successive queries (usually a threshold on a single feature - known as a univariate relational test), with each response leading to a following query until a state is determined[123, §7.2]. In this way, the tree uses different sets of features to classify different states (or portions thereof). In the case of an unbalanced tree, some classifications are made with far fewer decisions (and therefore far less energy) than others. Overall, the desire for hierarchical activation requires a hierarchical classifier. Similar arguments have been made in both medical[76, §16] and general[121] contexts.

This structure provides three key benefits. Categorical (as opposed to nominal) data can be handled directly as a single decision in the tree, rather than having to be assigned an arbitrary value and mapped onto an axis. There are a number of sensors that provide such output, either directly (such as tilt switches) or through internal thresholding (IR rangefinders). Also, the recursive partitioning process can often lead to a more accurate subdivision of the input space into various states than most single-function classifiers (such as support vector machines). Finally, the recursive structure makes for a fairly inexpensive classifier suitable for embedded usage. The average computational complexity of classification is $O(\log n)$ comparisons and the space complexity is $O(n)$(for n training examples with d features each)[27, §8.3].

Support vector machines (SVM) are often used as the classifiers in human application domains. SVMs perform classification by splitting the feature space into two classes along a hyperplane[123, §5.4]. This split can either be a direct linear mapping or (more commonly) one of a number of non-linear kernels applied to the input features. The examples closest to the hyperplane are known as the support vectors and define the classifier and the robustness thereof (based on their distance from the hyperplane). This algorithm is also typically much more expensive than decision trees, with a computational complexity of $O(d \log n)$ multiplications and the space complexity is $O(d \log n)$.

The key drawback of this algorithm is that access to all features is needed to make any decision. One possible workaround would be to redefine the classes such that each is either a state decision or a pointer to another classifier. This would allow for the creation of a series of classifiers using different subsets of the sensors, each with different power usage, which should result in reduced power usage (through the hierarchical nature, as above). However, the system would not be as nimble as a decision tree, due to the use of multiple sensors in each sub-classifier[7]. More importantly, it is unclear how to create and order these sub-classifiers, since SVMs were not designed to create a solution space capable of such subdivision. This suggests the intriguing solution of making the query at each node of the decision tree an SVM, though it is unclear whether the benefits (if any) would be worth the increase in complexity.

The key difference between these algorithms is their ability to cover the feature space. Decision trees create a sequence of subdivisions perpendicular to the feature axes, allowing

[7]The solutions from an SVM with only a single input would be a subset of the solutions from a similar classification tree since only a single cut is possible.

them to classify disparate sections as part of the same class[8]. SVMs define a single contiguous space as one class with all the exterior labelled as the other class. The boundary of this space is most often either linear or Gaussian - regardless, it is not bound to the feature axes. Therefore, SVMs can oftentimes have much greater descriptive power. This limitation of decision trees can be somewhat 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. By contrast, state spaces where diagonal cuts are necessary (as might be obtained if the gait shoe attachment were rotated 45 degrees) will be difficult for a decision tree to handle.

While not discussed in depth, it should be noted that the other classifiers commonly used in this application space - such as neural networks, hidden Markov models (HMM) and k-nearest neighbours - suffer from the same problem of lack of separability described above for SVMs. One distinguishing feature worth discussing is the time-dependance of state transition-based approaches, such as HMMs[99], which is not present in other techniques. These techniques are based on tracking the internal state of a system and the transitions between them. Each state is associated with an output probability density for the observables of the system, and a transition probability to other states. These techniques allow for the encoding of the probability of transition between certain states as more or less likely than other transitions. The seminal application of this approach is in speech recognition, where certain phonemes are far more likely to occur given the previous phonemes. For example, in a wearable gait application, we note that there are three subsets of states which can only transition between each other: non-ambulatory states, level gait and other ambulatory motions[9]. Such limitations could either be extracted from the long-term data stream or hand-coded by the application designer. While this could be quite beneficial in a number of applications, there would still be no good way of tiering the sensor usage for detection. Therefore, to decrease net power usage, the increase in accuracy would have to be achieved for less energy than the response to the interesting state would consume in the marginal

[8]This applies to standard univariate decision trees, as used in this work. It is worth noting that none of the major classification tree packages incorporate training for multivariate nodes and Breiman[14] has shown that they do not increase accuracy.

[9]It is assumed that transitions between, for instance, ascending stairs and shuffling are very rare.

false positives. This particular analysis is more application specific than we wish to make this framework, as it requires misclassification costs (in energy units, no less) to decide between two classifiers. We further note that, in comparison to SVMs (which are not strictly comparable to HMMs), our tree based classifier will give equivalent or better performance with respect to energy usage (section 7.3.2).

5.2.2 Priors and Costs

All things not being equal, we briefly discuss the usage of costs and prior probabilities in our algorithms. In this work, three different costs are specifically considered (for a comprehensive list of costs, see [122]):

Loss — Relative disinclination to predict one class when it is another. Often referred to as misclassification cost.

Test — Relative cost to acquire a feature used in the classifier. Broken down into the cost of acquiring the raw data and the cost of calculating the feature therefrom.

Time — Latency between state changes and recognition thereof.

Costs are used to alter the relative weight assigned to various states when making decisions during the classification process. Test costs are used to alter the selection of one feature over another during the construction of the tree. Loss is used to alter the assignment of a node to one class or the other. Time cost could be used to adjust the pruning of the tree (see section 8.1.3).

The test cost itself requires some definitions. We will use the notation introduced on page 17. Firstly,

$$TC = (TC_s + TC_f)/(t_{\textsc{Cyc}})$$

where TC is the test cost (in units of power), TC_s is the energy used to collect a sample, TC_f is the energy used to calculate a single feature based on that sample. We break this down further by considering the sensor cost and the feature cost. To whit:

$$TC_s = E_{S_i,\textsc{Wake}} + t_{\mu\text{P},\textsc{Adc}} P_{S_i,\textsc{On}} + E_{\mu\text{P},\textsc{Adc}} \tag{5.1}$$

i.e. the sum of the energy to wake the sensor and the energy to sample the ADC and power the sensor during the collection operation. Note that the power drawn by the sensor during wake up is not assumed to be the same as that when it is active. If the time to turn on the sensor and sample is greater than the sampling rate ($(t_{S_i,\text{WAKE}} + t_{\mu\text{P,ADC}}) > t_{\text{CYC}}$), then we replace equation 5.1 with:

$$TC_s = P_{S_i,\text{ON}} t_{\text{CYC}} + E_{\mu\text{P,ADC}}$$

since the sensor can no longer be duty-cycled.

Similarly, we define:

$$TC_f = (\#Ops) P_{\mu\text{P,ON}} t_{\mu\text{P,OP}}$$

where $\#Ops$ is the number of operations necessary to compute the feature. The above formulae are written such that there are no derived quantities. All of the values are available directly from the data sheets (with the exception of t_{CYC}). Note that in almost every case $TC_s \gg TC_f$ and therefore $TC \cong TC_s$. Numerical evaluation of these formulae for the sample application can be found in section 7.2.

While time and test costs are well defined, the misclassification cost is almost by definition ill-defined. In most cases, it will represent the designer's internal trade-offs between incorrectly recognizing a state (false positive) and missing it (false negative). For the case where power is the key criterion, this forces a comparison between the incomparable values of the power wasted by responding to a false positive and the data loss of a missed detection.

Prior probabilities represent our knowledge about the relative frequency of various states and are mathematically similar to loss. In the case where the cost of misclassifying a state has equal cost regardless of what it is misclassified as, loss and priors are indistinguishable (which is always the case for a two-state system). For the applications suggested in section 1.3, priors are best obtained from knowledge and/or studies of the underlying system. Because of our interest in infrequent states, the prior based on the long-term data streams are likely to have a large error due to the small number of representative segments for each state (the scripted data stream is, of course, useless for this purpose).

5.2.3 Overview of Decision Tree Construction

The construction of a classification tree is a straight-forward recursive algorithm which is effectively unchanged from those codified by Breiman[15] and Quinlan[97]. This technique, known as top-down induction of decision trees (TDIDT), is given below. We assume a univariate relational test and binary splits.

1. Begin at the root node of the tree with all examples.
2. For each feature, determine all the possible splits (thresholds).
3. For each split, calculate the value of the splitting criterion. Roughly, this value will be greater for purer splits (*i.e.* those where the split has skewed the distribution of examples on each side towards different states).
4. Execute the split with the highest value:
   (a) Set the query for the node to the feature and threshold of this split.
   (b) Branch to create two new nodes, one containing the examples above the threshold, the other with those below.
5. For each new node:
   (a) If it is pure or meets a stopping criterion (see below), stop and set this node as a leaf.
   (b) Otherwise, apply steps 2–4 to this node.

Further implementation details are available in the above references. A survey of splitting criteria is found in [105].

Issues Related to Classification with Test Costs

Overall, our goal is to make decisions for the minimal test cost for a given accuracy, rather than simply making the most accurate decisions. Therefore, a number of new issues arise with respect to the choice of algorithms.

The most important restriction is with regards to meta-algorithms which make use of a combination of individual classifiers. Bagging[13] increases the accuracy of decision tree classifiers by building a large number of trees using different training and testing sets and using the majority vote thereof to make a final decision. AdaBoost[39] acts recursively, biasing each subsequent classifier constructed towards the incorrectly classified examples of the previous ones. In each case, the structure of the decision trees change throughout the process, until it is likely that the number of sensors at the root of at least one tree will be quite large. At this point, any benefits from hierarchical activation will be lost.

One possibility for reducing the average test cost of classification is to alter the form of the decision trees itself. Most TDIDT algorithms attempt to grow balanced trees, which can require a large number of tests to make any decision. Trees with a higher ratio of leaves to nodes - *i.e.* those which make more decisions more quickly - may achieve similar accuracy for lower test cost. The most extreme case of such a decision tree is known as a decision list, where each node connects to one node and one leaf (with the exception of the final node). A possible technique for building these structures in a top-down fashion is considered below.

Finally, pruning is normally used to solely increase the generality of the classifier. As such, pruning algorithms have been labelled as biased towards overpruning or underpruning[32]. Since pruning, in general, tends to reduce the test cost of the decision tree, we will want to choose a technique which gives smaller trees, even at the possible price of reduced accuracy. However, this loss should be minimal since the deepest nodes, by definition, include only a small number of examples.

5.2.4 Decision Tree Splitting Criteria

Because of the concerns given above regarding the availability of the priors and costs, it is preferred to use a splitting criterion which is robust to variation in these values - *i.e.* one based solely on the enumeration of the training examples (cardinality). The goal is to grow as general a tree as possible and then extract a final version through pruning and node assignment using the eventual values for prior probabilities and misclassification costs.

The receiver operating characteristic (ROC) is one way to codify this concept of generality. For a two-class system, the ROC is a plot of the true positive rate against the false positive rate. Values nearer to the top left corner are better, and $(0,0)$ and $(1,1)$ can be obtained trivially. For a trained decision tree, the ROC can be generated simply by calculating the true and false positive rates for each possible assignment of the leaf nodes. The area under the ROC curve (AUC) is a good metric for the quality of a classifier[11, 51].

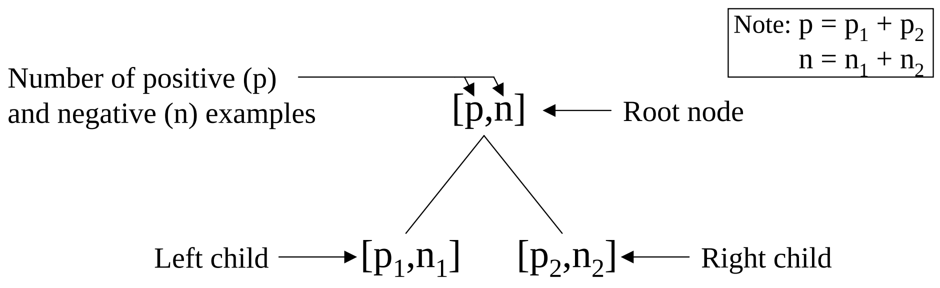


Figure 5-2: Classification tree split

A splitting criterion which maximizes the AUC of each split is adopted from [36]. For the split shown in figure 5-2, this criterion gives[10]:

$$C_{AUC}(s) = \frac{1}{2}\left(\frac{p_1}{p} - \frac{n_1}{n} + 1\right) \tag{5.2}$$

where s is a specific split (*i.e.* the p_i and n_i values) and higher values of $C(s)$ are better. This criterion has several beneficial properties. It is based solely on cardinality, as desired. Further, evenly sized splits (relative to the population at the parent node) are generally better than smaller, purer splits (which can lead to overfitting). Most importantly, it has been shown that the performance of this classifier does not degrade significantly even with a large change in priors, which is not the case for those constructed using other splitting criteria[36]. Therefore, a general (*i.e.* non-specific) criterion not only produces robust classifiers for a large range of weights and misclassification costs, but can also be used to argue for the generality of the techniques presented.

We note two concerns with this approach. The first is that it only applies to two class systems[11]. This can be easily accommodated by setting the state of interest as the positive

[10]This assumes (*wlog*) that the left branch has a higher positive accuracy (*i.e.* $\frac{p_1}{p_1+n_1} > \frac{p_2}{p_2+n_2}$)

[11]While Hand and Till[47] have extended ROC analysis to systems with greater than two dimensions, this has not been shown to produce a good splitting criterion.

class and combining the rest as the negative class. This does not prevent the states from being individually tracked, it merely means that the tree will not attempt to separate them. This restriction is mitigated by the claim that there is usually only one interesting state per application (or subset of the application) and the fact that a tree separating two classes will never be larger than one separating more. Second, while tree construction is based solely on cardinality, specific values for the priors and misclassification costs will still be necessary to set the assignment of the leaf nodes and to prune the tree.

We consider two other popular splitting criteria in this context. C4.5, another classification tree implementation by Quinlan[98], uses an information (entropy) based criterion:

$$C_{info}(s) = \sum_{i \,\epsilon\, child} P(i) \sum_{j=+,-} P(j|i) \log P(j|i) \tag{5.3}$$

where + and - refer to the positive and negative classes, respectively. This gives:

$$C_{info}(s) = \frac{N_1}{N}\left(\frac{p_1}{N_1}\log(\frac{p_1}{N_1}) + \frac{n_1}{N_1}\log(\frac{n_1}{N_1})\right) + \frac{N_2}{N}\left(\frac{p_2}{N_2}\log(\frac{p_2}{N_2}) + \frac{n_2}{N_2}\log(\frac{n_2}{N_2})\right) \tag{5.4}$$

for the case above (figure 5-2), where $N = p + n$ and $N_i = p_i + n_i$. The split with the maximum value is chosen. In the case of non-binary splits, C4.5 uses a function known as gain ratio, which is as above, except divided by the entropy of the split population (since equation 5.4 favours M-way splits based solely on larger M). The Gini criterion used in the CART algorithms[15] is a simplified version of the entropy of the split:

$$C_{Gini}(s) = \sum_{i \,\epsilon\, child} P(i)P(+|i)P(-|i) \tag{5.5}$$

For the case above, we have:

$$C_{Gini}(s) = \frac{N_1}{N}\left(\frac{p_1}{N_1}\frac{n_1}{N_1}\right) + \frac{N_2}{N}\left(\frac{p_2}{N_2}\frac{n_2}{N_2}\right) \tag{5.6}$$

The split with the smallest value is chosen. As seen in figure 5-3, the Gini criterion and the information criterion are nearly identical, with the exception of the much lower cost of execution for Gini.

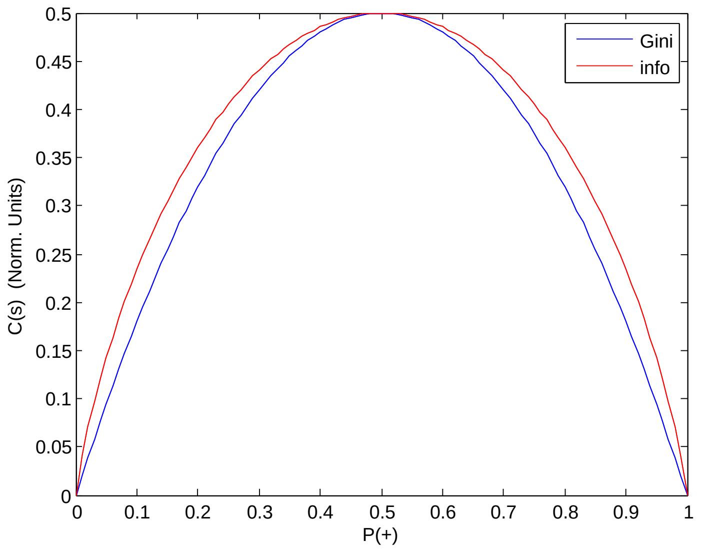


Figure 5-3: Comparison of the Gini and information splitting criteria

Both of these criteria are designed to minimize the error (or risk, if weights are taken into account) of the decision tree. Therefore, they are sometimes biased away from small classes towards more populous ones, such that a split incorporating noisy examples from a high probability state will be chosen over a split based on accurate examples from a rare state. This can be alleviated by setting weights or artificially generating more examples of uncommon states[19], but these techniques tend to be finicky at best.

To examine whether it is possible to achieve good performance while forcing the decision tree into a list structure (one side of each split is a leaf), we will construct a modification of the Gini criterion. At each node, we will pick the split which maximizes the criterion for a single side, with the requirement that the chosen side has a lower impurity than the current node and that it contain at least 5% of the examples (to avoid small pure nodes dominating the tree). The maximal side is set as a leaf, and splitting continues on the other branch. We use this technique because the current induction algorithms for decision lists (such as [22]) are bottom up, which prevents the use of test cost, and are designed to split on combinations of binary variables.

While a wide variety of splitting criteria are discussed in the literature, studies show that the choice thereof has little to no effect on accuracy[77]. Therefore, any criterion which gives good results with the test cost weighting described below is acceptable.

Test cost weighting

To be able to take test cost into account when building the tree, the splitting criterion must be a function thereof. Therefore, we alter it as follows:

$$C'(s) = \frac{C(s)}{f(TC, W)} \tag{5.7}$$

where $f(\cdot)$ is some function of the test cost and a parameter W. The intent is to reorder $C'(s)$, relative to $C(s)$, based on the test cost. W controls the extent of the reordering, with $W = 0$ having no effect and $W = \infty$ forcing only the sensor(s) with the lowest test cost to be chosen.

Further, the structure of the decision tree needs to be taken into account. Since this is a sequential classifier, once a feature has been used to make a decision, the sensor from which that feature was calculated will then be available to all children of that node for a discounted cost. Specifically, if the initial use is at a frequency f_1, the new cost to sample the sensor at a second frequency f_2 is:

$$TC'_s = \begin{cases} 0 & f_1 \geq f_2 \\ TC_{s(f_2)} - TC_{s(f_1)} & f_1 < f_2 \end{cases} \tag{5.8}$$

The feature cost is unchanged. This form makes both intuitive and analytic sense. 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. An increase in sampling rate should only bear the cost of the increased power usage[12]. Note that this form also puts all activated sensors on equal footing, regardless of their power usage. Sensors which share a common cost (such as an amplifier) can also be easily accommodated. Any sensor used following one with which it shares a cost is simply discounted by that amount.

We begin with a generalized multiplicative weighting function:

$$f(TC, W) = (\alpha + \beta(TC))^W \tag{5.9}$$

[12]Again, this refers to the special case where downsampling is trivial (*i.e.* if the new frequency is an integral divisor of the old frequency). This will always be the case for the sampling frequencies chosen in section 5.1.3.

The parameters of this form will be examined in detail. We first set $\alpha = 1$ without loss of generality, since it can simply be divided out without altering the ordering of the function. The value $\alpha = 0$ is not allowed as it would make f undefined for activated sensors ($TC = 0$). In fact, $f(0, W) = 1$ for $\alpha = 0$, such that splits involving activated sensors are discriminated solely on the original criterion $C(s)$. If, for some reason, $TC \neq 0$ for activated sensors, this value could just be subsumed into α and then divided out.

Choosing a value for β is somewhat more complicated and tied to the individual values of the test cost. We define $\gamma \equiv \beta \min(TC)$ for ease of discussion. For $\gamma = 10 \gg 1$, there are effectively two separate operating ranges for f: the first for activated sensors, where it has no effect whatsoever, and the second for unused sensors, where it has a relative value proportional to the power usage of the sensors. It is also possible to set $\gamma \approx 1$, which will skew the function to give lower relative cost to higher power sensors. However, it is unclear whether or when this is a desirable property. Therefore, it is best to keep $\gamma \geq 10$ unless obvious flaws are found in the resultant trees (such as the insistence on using slightly lower power but far less discriminatory sensors).

Finally, the parameter W adjusts the relative importance of power in the classifier construction. 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 the same tree 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 to be able to grow a population of different trees by varying W. The exact relationship between W and total test cost is not examined.

We consider the useful range for W. Since the intent of the weighting function is to reorder the original splitting criterion, the value of the minimum value of f for $TC \neq 0$ cannot exceed the dynamic range of $C(s)$. Specifically,

$$\frac{\max(C(s))}{\min(C(s))} > \min_{TC \neq 0}(f(TC, W)) \tag{5.10}$$

If this does not hold, then no unused sensor will ever be activated since even in the case of a perfect split (whose definition varies with the criterion) the value of $C'(s)$ cannot be greater than that of the worst possible split of an activated sensor. The AUC-based criterion ranges between 0.5 (useless) and 1 (perfect), and therefore:

$$\begin{aligned} \frac{1}{0.5} &> (1+\gamma)^W \\ \Rightarrow \log(2) &> W\log(1+\gamma) \\ \Rightarrow W &< \frac{\log(2)}{\log(11)} \cong 0.29 \end{aligned} \tag{5.11}$$

for $\gamma = 10$ as above. Redefining $C'_{Gini} = 1 - 2C_{Gini}$ gives this function the same range as above and hence the same maximum value of W.

Several other weighting function have been used in the literature. The functions of Tan[117]:

$$C'(s) = \frac{[C(s)]^2}{TC} \tag{5.12}$$

and Norton[84]:

$$C'(s) = \frac{C(s)}{TC} \tag{5.13}$$

are equivalent to ours for $\alpha = 0$, $\beta = 1$ and $W = 0.5$ and 1, respectively. Obviously, neither function allows sensor costs to be set to zero in the case of repeated use of the feature since this would make $C'(s)$ undefined. Further, their choice of parameters (α, β, W) are not justified beyond unsupported claims of optimality[13]. Núñez[85] models the system as a Shannon transmission line[108] and finds a signal to noise ratio of:

$$C'(s) = \frac{2^{C(s)} - 1}{(1+TC)^W} \tag{5.14}$$

where $C(s)$ is defined as in equation 5.4 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. Finally, note that in all three cases, the cost function is multiplicative.

[13]In this case, defined as growing the tree which most accurately classifies an independent test set, possibly for lower average classification cost.

5.2.5 Decision Tree Pruning

Classification trees are grown until they are pure (*i.e.* the leaf nodes contain only examples of a single class) or until some stopping condition is met. In general, this tends to lead to trees which overfit the training data and then need to be pruned back[123]. Two types of pruning are considered: pre-pruning, or stopping criteria, and post-pruning.

There are a number of stopping criteria in the literature. The most commonly used is the imposition of a minimum node size. Any smaller node is set as a leaf. This acts as a check against expending effort to distinguish between a small number of likely noisy examples and on the possibility that two examples from different classes are indistinguishable (which only tends to occur with categorical data). A chi-square test on the significance of the split is sometimes used, though it is somewhat redundant with the first criterion (since small nodes rarely pass) and can reject some important cases[32]. Minimum description length pre-pruning looks for the global minimum of:

$$\alpha \cdot size + \sum_{leaves} H \tag{5.15}$$

where *size* is a measure of the size of the tree (often total nodes or leaf nodes) and H is the entropy of the node. However, relationships between α and the accuracy of the classifier are difficult to establish[27]. In general, pre-pruning is not considered to be beneficial and should be avoided[123]. In this implementation, the only pre-pruning used is a limitation on splitting nodes with fewer than 5 examples, as recommended in [37].

Post-pruning is the process of taking a fully grown tree and replacing subtrees by their root node to improve the performance. All common techniques are designed to maximize the accuracy or minimize the risk (error rate times loss) and therefore require both priors and misclassification costs. Post-pruning is beneficial for two reasons. Numerically, it removes nodes which do not alter the accuracy and those which degrade the accuracy since they are based on idiosyncracies of the training set. Analytically, it brings the consideration of the key metric of classifier performance, accuracy, into consideration for the first time. While it may be argued that splitting based on the desired global metric is not the best way to construct a tree classifier (in fact, no common algorithm does so[105]), that metric is clearly the best way to prune a tree.

There are a large number of different pruning algorithms[32], two of the most common of which are considered below. The C4.5 package[98] uses a technique known as error-based pruning. The algorithm treats each node as a binomial random variable, and replaces its current error with the upper bound of the error for a fixed confidence interval (usually 25%). A bottom-up traversal is then performed over all nodes, comparing the upper bound of the error of each interior node with the subtree beneath it. If the former is smaller, the subtree is removed. The comparison is sometimes extended to include the most populous child of the interior node, with that child (and its subtree) moved into the position of the interior node if it has the minimal error. However, this addition is not appropriate for our purposes, as the test cost weighting during tree construction is predicated on the nodes above. The algorithm uses the training set populations to estimate the upper bound of the error, and therefore does not require a separate pruning set. However, assumptions necessary for the binomial modelling (specifically independence and sufficiently large nodes) do not hold, as argued by Esposito[32] and acknowledged by Quinlan[98]. Further, while the confidence interval can technically be changed, this is rarely done in practise, often leading to underpruned trees.

The CART package[15] uses an algorithm known as cost-complexity pruning (CCP). It uses effectively the same formula as the minimum description length (equation 5.15), where the entropy is replaced by the risk and $size$ is set to the number of leaf nodes. An order of pruning operations is created by comparing the cost-complexity of each internal node with the sum of the cost-complexity for all the leaves below it and finding the value of α necessary to make them equal. The subtree with the lowest value of α is pruned first. The process is then repeated on the new tree until only the root node remains. Given this ordering, a level of pruning which trades error rate against depth can be chosen. In most cases, the minimum of the curve is found and then the smallest tree (in the ordering) with an error rate within one standard error of the minima is chosen. This procedure is most often referred to as CCP-1SE. This technique has a tendency to overprune[32], which, in fact, is desired in this case (section 5.2.3). This algorithm is most often run using cross-validation, and therefore does not require that a separate pruning set be extracted from the available examples, though a holdout training set can also used. While it has the drawback that it does not test all possible subtrees, it has been shown[15] that the generated subtrees are optimal with respect to out of sample error for their size.

5.3 Examinations of Features

In this section, we examine the feature algorithms chosen in section 5.1.3 to show that they have the consistency (on a class-by-class basis) desired for classification. This analysis uses the data streams collected in section 7.1, though specific details thereof are not necessary to understand this section.

The tests in this section are done using the data for the y-axis (upward facing) accelerometer, which proved best for visual analysis for the majority of the motions tracked[14]. Results for the other sensors are qualitatively similar. For reference, a portion of the data captured from the accelerometer is shown in figure 5-4. The red boxes surrounding portions of the data represent the extent of different motions. This segmentation was done by hand using a video recording as a reference for the annotation. The motion type itself is not labelled on the figure because of legibility concerns. They are: ascend stairs, turn, descend stairs.

[14]This is not to be confused with the ability to disambiguate states.

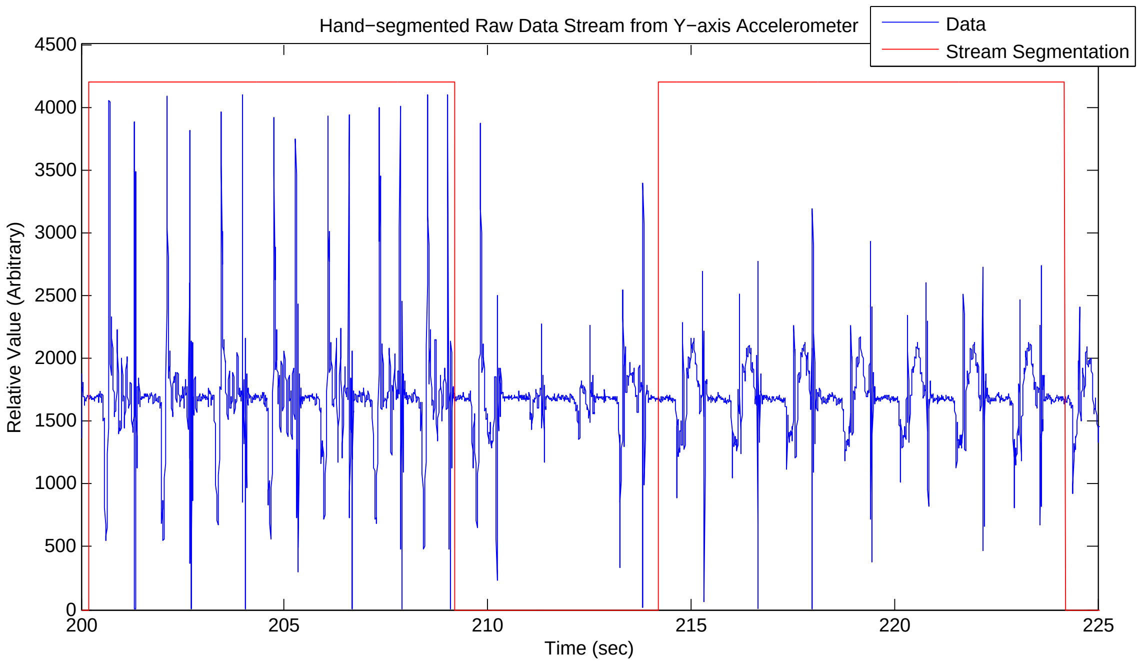


Figure 5-4: Portion of sample data stream with segmentation

5.3.1 Consistency over Window Size

The windowed mean and variance of the y-axis accelerometer over the training data stream is shown in figure 5-5. Again, the red boxes indicate the segmentation of the stream into various motions based on the video record. The features were only calculated within the labelled areas, hence the gaps in the stream. Note that the values are fairly consistent across individual motions, and some motions can be differentiated based solely on these features.

The range of window sizes to consider can be determined in two ways. First, visual examination of the data can often provide an accurate starting value for the search. Second, studies in the field may directly provide the information desired. In this case, the cycle time appears to be roughly one second. Also, Finley and Cody[38] studied the walking rates of urban pedestrians and found an average of 55 steps/min (or 1.1 sec/step). Using the second value as a guide, window sizes examined will centre on 224 points (at 200 Hz sampling) and range from 192 points to 256 points in steps of 8 (to provide for integer window sizes for 8 times downsampling to 25 Hz). Because all of the sensors capture a single phenomenon, it is reasonable, but not necessary, for them to share a common window size.

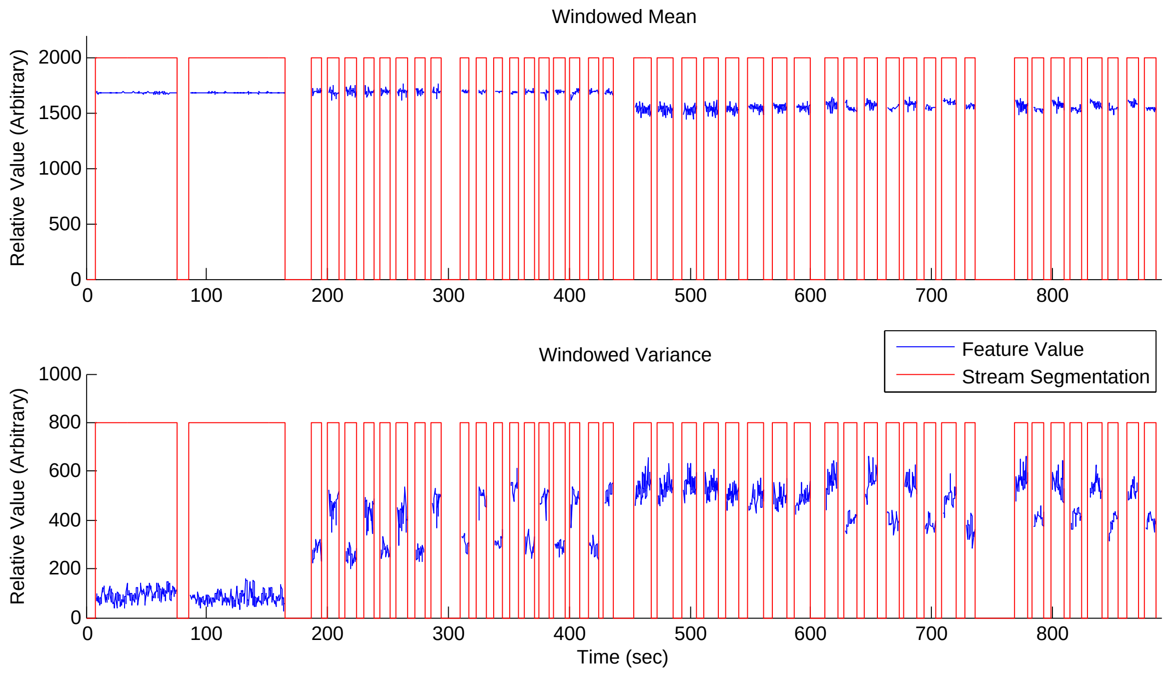


Figure 5-5: The windowed mean and variance features (1.2 sec window size)

In the case of sensors capturing orthogonal phenomena, different window sizes will most likely be necessary.

Figure 5-6 organizes this information in the form of four plots, with the features now separated by class. The average of the mean and variance features show that the values are fairly consistent across the range of windows considered. This suggests that inaccuracies in the step frequency (at least within the range considered) will produce at most small differences in these variables over large sets and allows for the direct comparison of the standard deviations of the mean and variance features. Since these values directly effect separability, the minima will be used to choose the best window size. For the mean plots, units are arbitrary (derived from ADC counts). For the standard deviation plots, each curve is scaled such that its minimum value is equal to one. This places all of the minima along the x-axis and allows the relative quality change with window size to be read off the y-axis. Examination of the plots suggests that a window size of 240 points is the best choice for this data.

The windowed maximum and minimum over the sample data stream are shown in figure 5-7 and figure 5-8. These values proved inconsistent over the window size chosen, most

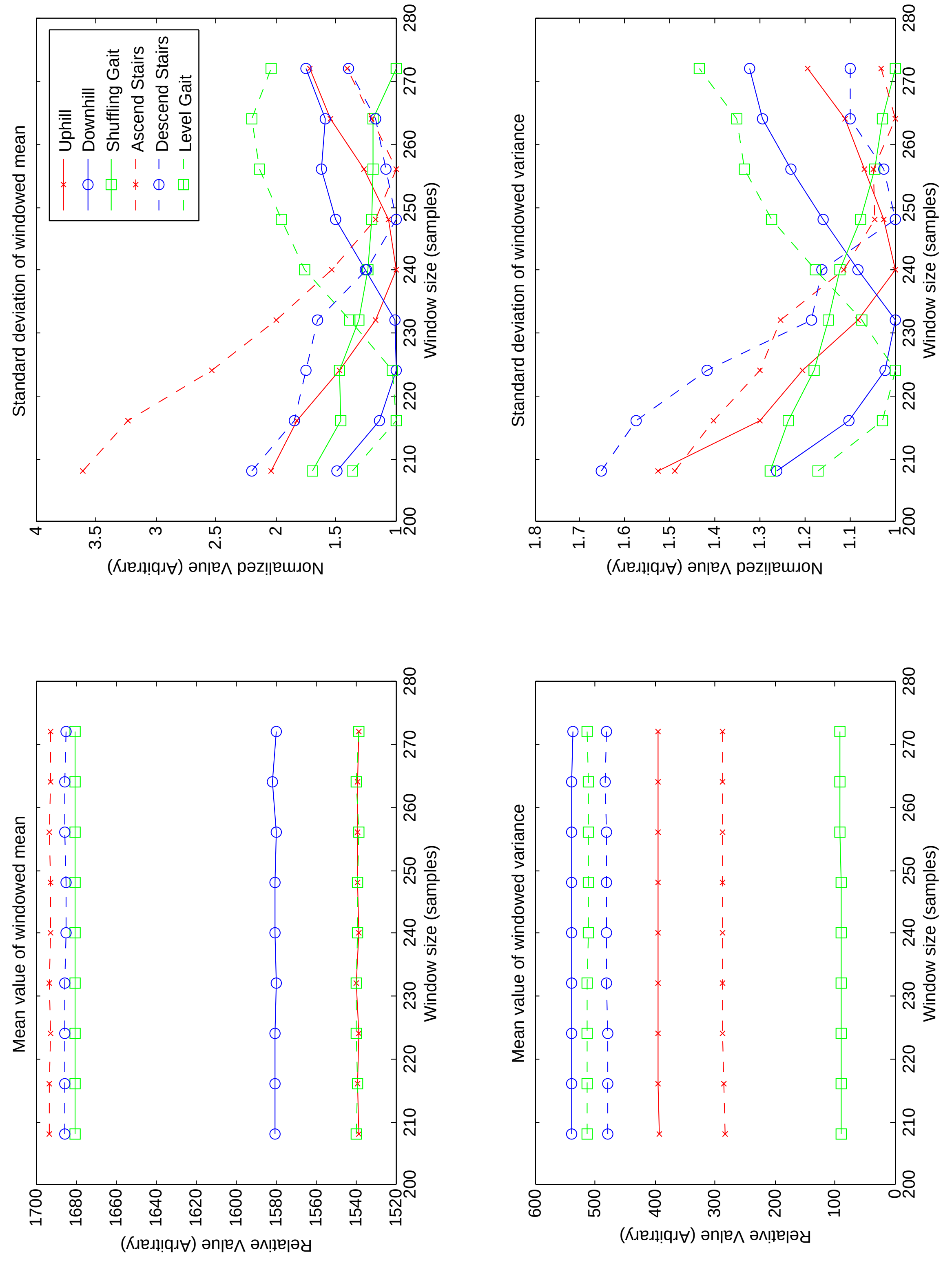


Figure 5-6: Effect of window size on mean and variance features

likely due to fast transients which were not completely captured by the hardware, and would have proven virtually useless for training the classifier. Replacing the maximum and minimum with the more robust 90th and 10th percentile (respectively) provided for much more consistent results. While this analysis suggests that analysis of the robust versions of mean and variance (termed clipped) would be fruitful, the increased complexity of calculation (since they necessitate knowledge of the variance) would be unacceptable. The use of fixed high and low thresholds beyond which data points are assumed to be outliers could be used, though it would have to be done on a sensor-by-sensor basis.

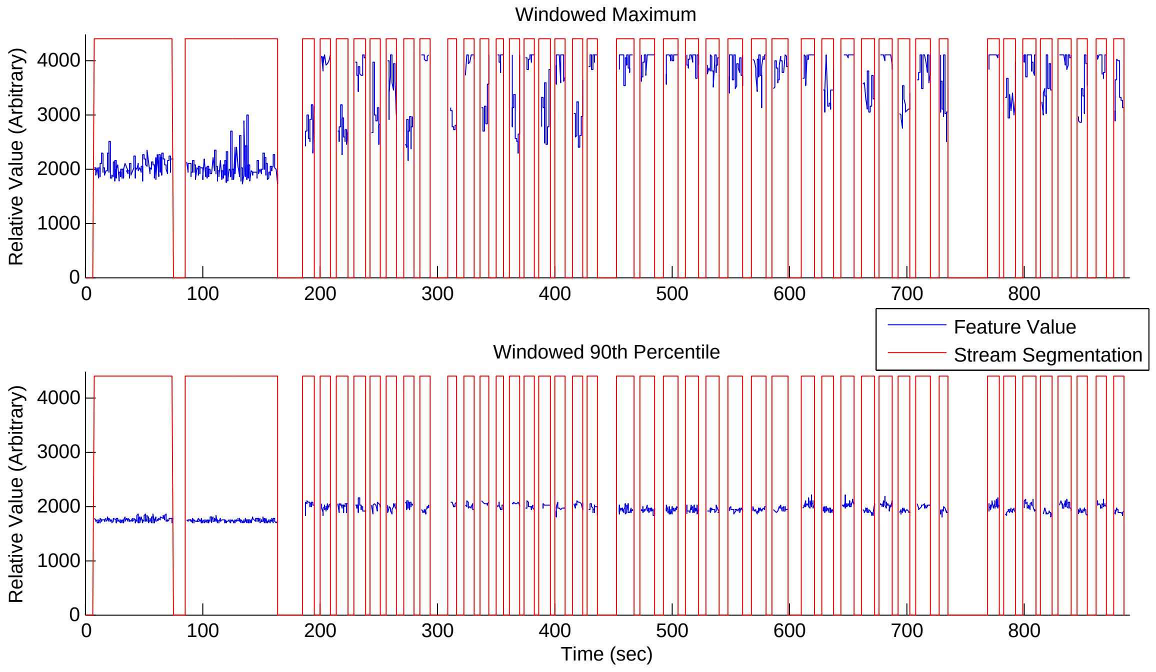


Figure 5-7: Comparison of windowed maximum and windowed 90th percentile features

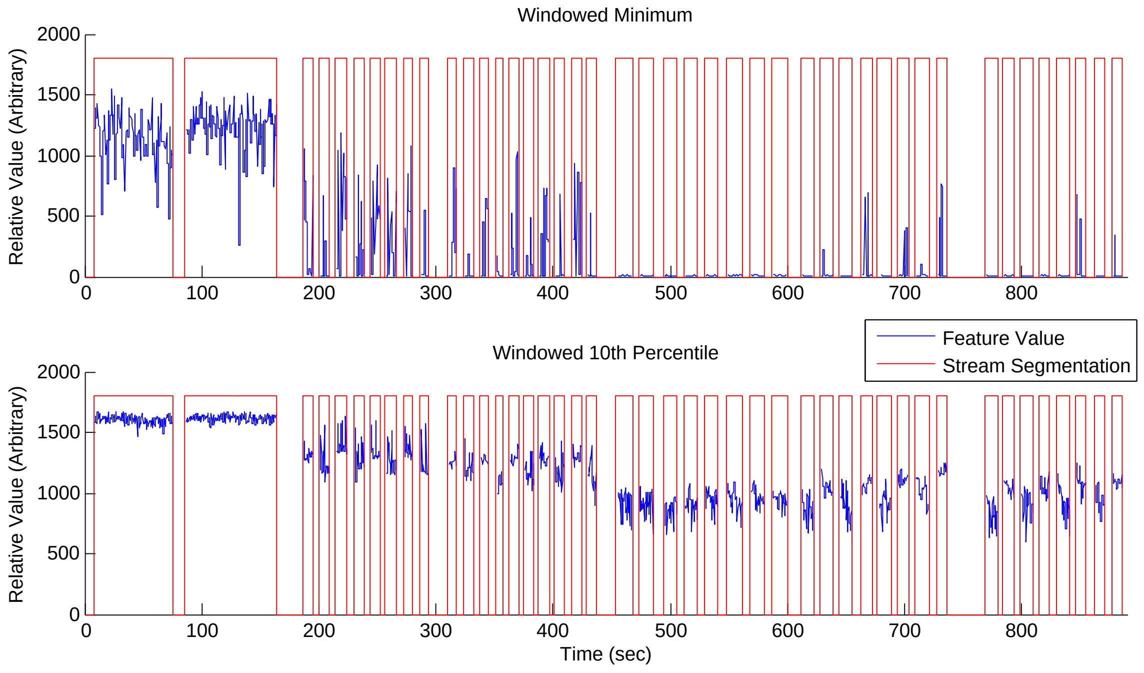


Figure 5-8: Comparison of windowed minimum and windowed 10th percentile features

5.3.2 Consistency over Frequency

Having selected a window size above, it is worthwhile to examine the variation of the features with respect to sampling frequency. This will give a sense of the tradeoff between the accuracy of the classifier (tied to class separability) and sensor power (tied to the sampling frequency). Figure 5-9 shows the mean and the standard deviation of the windowed mean and variance features with respect to frequency. The standard deviation plots are scaled as before. The mean of the features, as before, is effectively constant. The standard deviation begins by dropping sharply, but appears to be approaching an asymptote near 100 Hz. This suggests that features at this sampling frequency may be as effective at disambiguating states as those collected at 200 Hz.

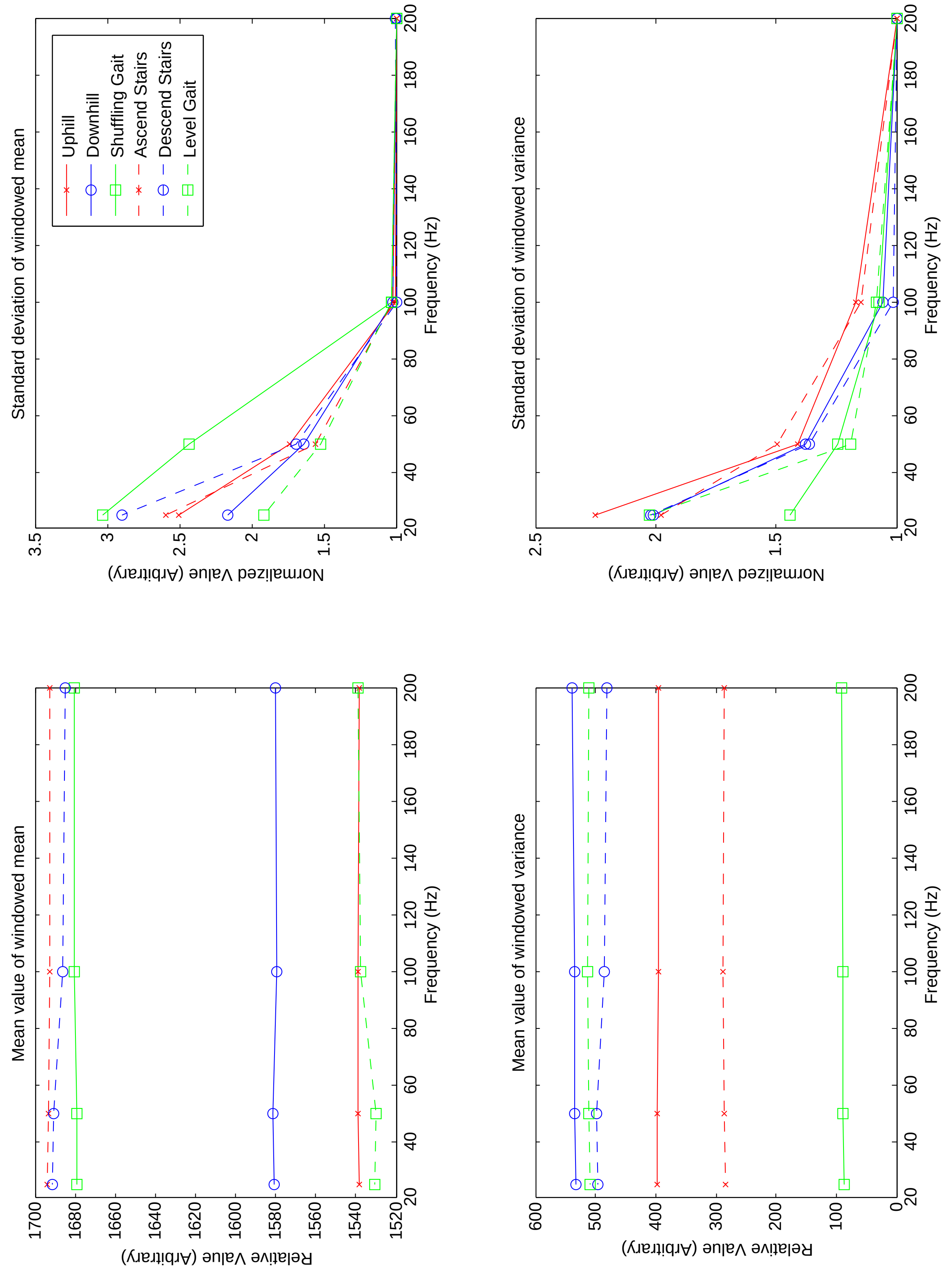


Figure 5-9: Effect of sampling frequency of mean and variance features

Name	Classes	Features With Non-Zero Cost	Features With Zero Cost	Trivial accuracy[a]
Pima Indians Diabetes Database (Pima)	2	2	6	0.65
BUPA Liver-disorders Database (BUPA)	2	6	0	0.51
Heart Disease Database - Cleveland (Clev)	2[b]	10	3	0.54

Table 5.1: Summary of data sets used for testing

[a]Can be achieved by always selecting the majority class
[b]The four subsets of the sick class were treated as a single class.

5.4 Examination of Test Cost Weighting

While altering the splitting criterion of a decision tree will, of course, alter the generated tree, it is far from clear exactly what the effect will be. In this section, we will examine how the accuracy, test cost and latency of classification trees varies with W for a variety of data sets. Tests are run with the Gini criterion (both normal and single-sided) and the AUC-based criterion. The data sets used are from the University of California Irvine (UCI) machine learning repository[82] with test costs provided in [121]. Table 5.1 details the relevant parameters of each set. All are from medical diagnostic domains, with test costs reflecting the financial cost of running blood tests. Testing was done using a modified version of the MATLAB implementation of CART (an overview of changes we made can be found in appendix C).

5.4.1 Discussion of Testing Methodology

We originally intended to use a standard decision tree construction methodology here. Specifically, the classifier was to be constructed using one of the splitting criteria to be examined and then pruned using 10-fold cross-validation cost-complexity pruning with the one standard error offset (CCP-1SE). Parameters could then be estimated from 10 loops of the cross-validation procedure, as recommended in [15, 123]. However, closer examination of the trees produced by the classifier suggested that there were some difficulties with this approach. Specifically, while only a small number of different trees were produced, the graphs of accuracy and test cost against W were smooth, suggesting a much larger population. The cause was instability in the cross-validation procedure, which is discussed below (in this case, the cause was numerous features with near equivalent descriptive ability).

One known potential difficulty with the growing of the decision trees is the instability of the top-down induction of decision trees themselves, as previously noted in the literature[15, §5.5.2][48, §9.2.4]. Such problems can arise under two circumstances. In the first, multiple features give splits of near maximum quality at a specific node. In the second, a subset of the features are correlated and therefore their splits are of similar quality (albeit not necessarily maximal) at most nodes. Slight deviations, caused by noise or sampling errors, will determine the feature chosen by the training algorithm, which then affects the evolution of the remainder of the tree. In most circumstances, this effect is not of great concern, since the collection of trees grown tends to have similar accuracy. However, when the test cost parameter is of importance, there may be large variations in the average cost of classification based on the order in which the features (and hence the sensors) are chosen. While it was expected that the test cost weighting of the splitting criterion would prevent this problem from occurring, the structure of the induction (and the data) meant that once certain sensors were chosen, the subtree beneath it was all but fixed. Further, the test cost weighting merely changed the ordering of the features, but did not prevent groupings of features of similar quality.

These difficulties are compounded by the structure of testing application (see section 7.1), which purposefully sampled from a large number of sensors measuring identical phenomena in different fashions. This leads to the high correlations described above. Also, the large time and expense of collecting and annotating human motion data leads to the use of smaller data sets, which are by definition noisier. Such sets usually call for the use of cross-validation procedures for pruning and error (and other parameter) estimation. The removal of a portion of the data from the training set leads to wide variability in the trees grown, which directly increases the standard error of the estimation. In fact, the effect may be to the extent that cross-validation itself is not as appropriate, since the stability and similarity assumptions of the procedure are no longer met.

A remedy to allow the use of cross-validation is not evident. Bagging (described earlier) is often used in such cases (and with decision trees in general), but is not applicable here because of the cost of execution (both in terms of information and processor time). Li[65] has proposed a technique for overcoming instability in TDIDT. However, it requires using a logical conjunction (or disjunction) of tests at nodes where two (or more) tests are similar enough that either could be chosen given small changes in the training population. While effective, this greatly increases the test cost of evaluating the tree, and is therefore unacceptable as well.

Hence, standard holdout training methods will be used. Specifically, 20% of the data will be held out for a test set, with the remaining 80% again divided 80/20 between a training set and a pruning set (CCP-1SE is used). Overall, this uses 64% of the data for growing the tree, 16% for pruning and 20% for testing. This holdout procedure is stratified - *i.e.* the proportion of the various classes in each subset is as close as possible to that of the full set. The disadvantage of this procedure compared to cross-validation is that a much smaller proportion of the data is used for calculation of the parameters of the classifier. We note also that using less data to train the trees leads to greater differences amongst the grown trees. However, this variation is of little concern as our goal is to produce a large population and select amongst them based on their parameters. To be able to do so, estimation errors must be reduced through the use of a large enough test set such that the three segments are each reasonably representative.

5.4.2 Results

For each of the three splitting criteria, a population of trees was constructed using the above holdout procedure. W was varied between 0, which trains a tree without consideration of test cost, and 0.3, which uses only the cheapest feature(s). The relationships between W and accuracy (error rate of the holdout set) and W and test cost (average classification cost of the holdout set) are most important, though the latency and depth of the tree are also of interest. The latency is the average number of different features used to determine the class. This corresponds roughly to how long it would take to classify an example if a fixed delay was associated with each new feature. The depth is the average number of comparisons made to determine the class, and therefore is a measure of the computation necessary to classify an example.

For the Pima database, the results are fairly clear (figure 5-10). The one-sided split is insensitive to W and can be ignored. The Gini and AUC splits are very similar, with perhaps sightly better performance by the Gini split. More important is the general progression of the operation of the classifiers as W is reduced. Using the Gini criterion produces four distinct trees with different operating points. The simplest tree (figure 5-11d) is the trivial tree, which simply chooses the most populous class and assigns it to all of the examples. As W is reduced, useful trees are grown, beginning with one made up of only free features - the patient's age and body mass index (figure 5-11c). As the test cost is reduced further, the most descriptive and expensive test - plasma glucose level - appears (figure 5-11b) and eventually moves to the root of the tree when cost is no longer taken into account (figure 5-11a). As indicated by the peaks in the latency and depth plots, for middling values of W, the algorithm constructs more accurate classifiers by using a greater number of cheaper tests in place of a single expensive one. For the AUC-based criterion, the only notable difference in the classifiers built is that the most accurate tree does not occur at $W = 0$. Since the tree growing algorithm is a top-down greedy process, being forced to choose less expensive tests allows the algorithm to find an overall more accurate classifier than if it was given free reign.

The results for the BUPA database (figure 5-12) are straight-forward. Again, the one-sided splitting criterion only returned a single unique tree. The Gini and AUC-based criteria constructed the same trees (with one exception), only for different values of W. Since multiple values of the weighting factor are used as a matter of course, this difference is not significant.

The results for the Cleveland heart disease database (figure 5-13) are very similar. We quickly note two points. The one-sided splitting criterion was able to find two different trees, though neither was optimal. The Gini criterion found one more tree than the AUC-based criterion, though in both cases the best results came from trees grown with higher values of the weighting factor.

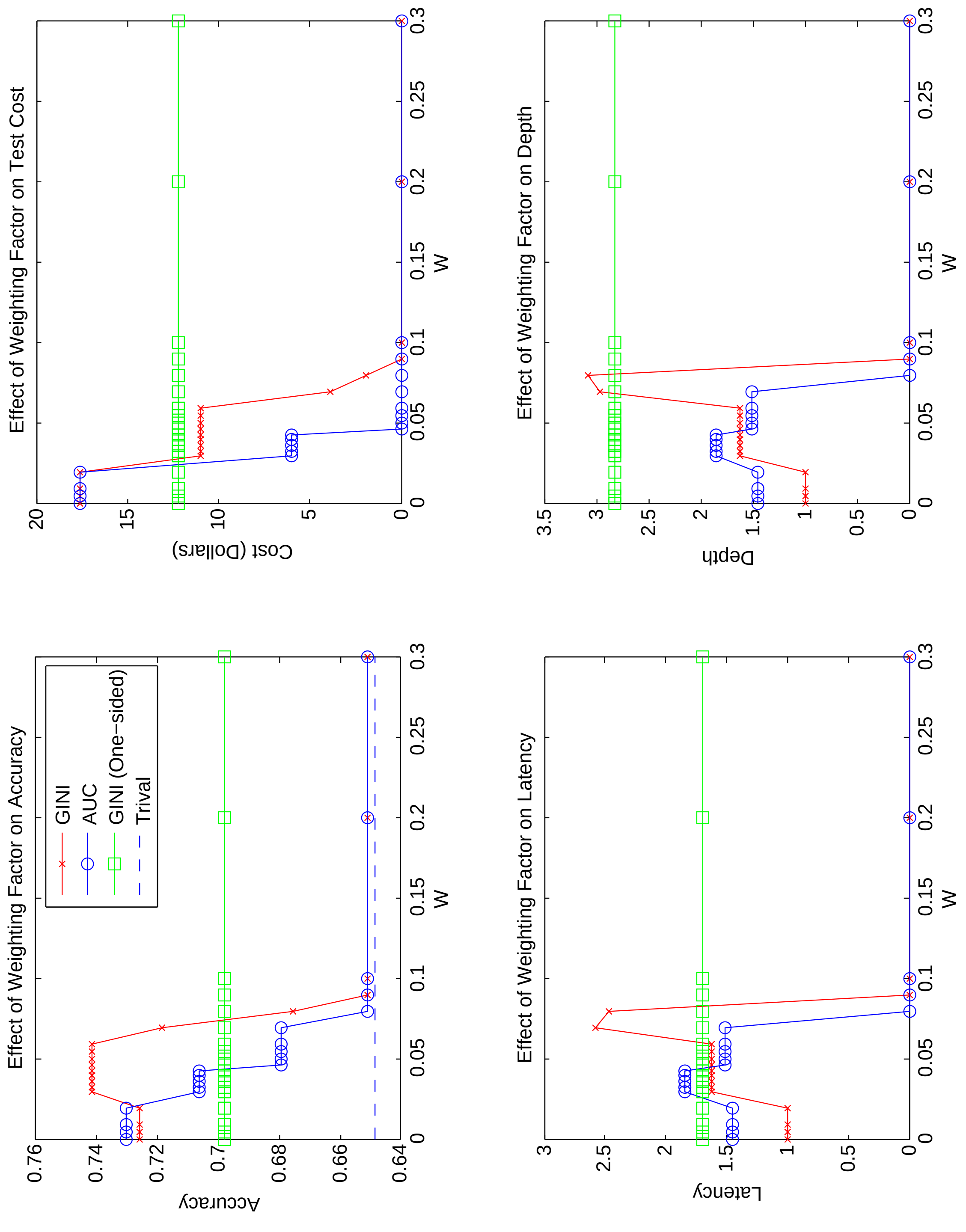
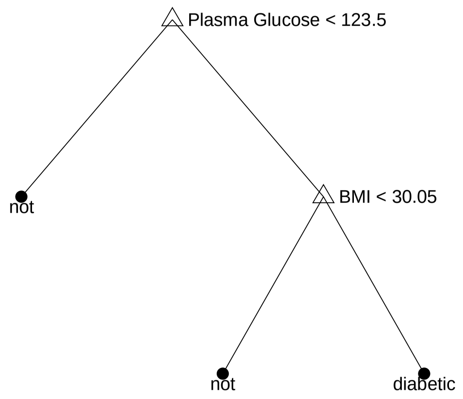
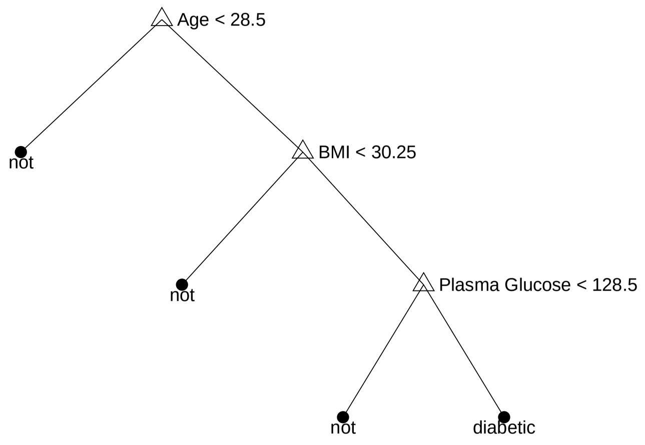


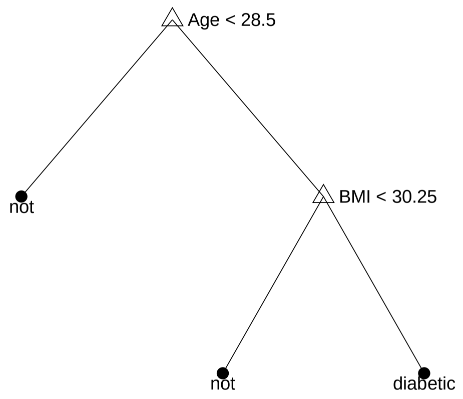
Figure 5-10: Effect of test cost weighting on classifier performance (Pima)



(a) W=0.00



(b) W=0.03



(c) W=0.05

(d) W=0.10

Figure 5-11: Sample classification trees for Pima database

A few general points can be made regarding these tests. First, there is no clear advantage to either the Gini or AUC-based criteria. However, the one-sided criterion is not responsive to changes in the weighting factor for these databases. While this may be caused by the low maximum accuracy of the classifiers for these problems, the criterion remains unproven and therefore will not be used in future tests. Second, there tends to be only a small number of trees grown for each database, suggesting that most features (or combinations thereof) are not particularly useful or are masked by other more useful features. While no general statement can be made, this does suggest that the tree growing algorithm is reasonably effective at feature selection (*e.g.* only three of the eight total features in the Pima database are ever used), in which case the plan to provide the classifier with data from a number of sensors of unknown utility has a limited risk of overtraining. Third, the core system is functioning as desired. Roughly, decision trees of increasing cost and accuracy are grown with decreasing W. In most cases, expensive but effective substructures within the tree progress towards the root node as the test cost weighting is reduced. This creates the desired population of trees, each at a different point in the power/accuracy plane.

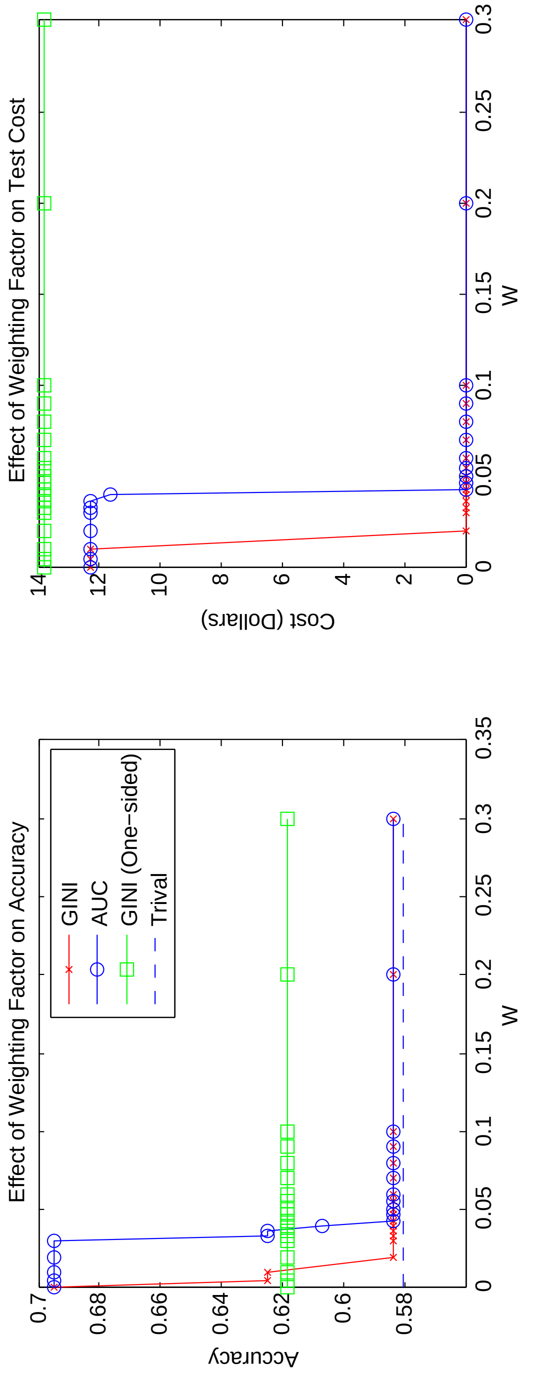


Figure 5-12: Effect of test cost weighting on classifier performance (BUPA)

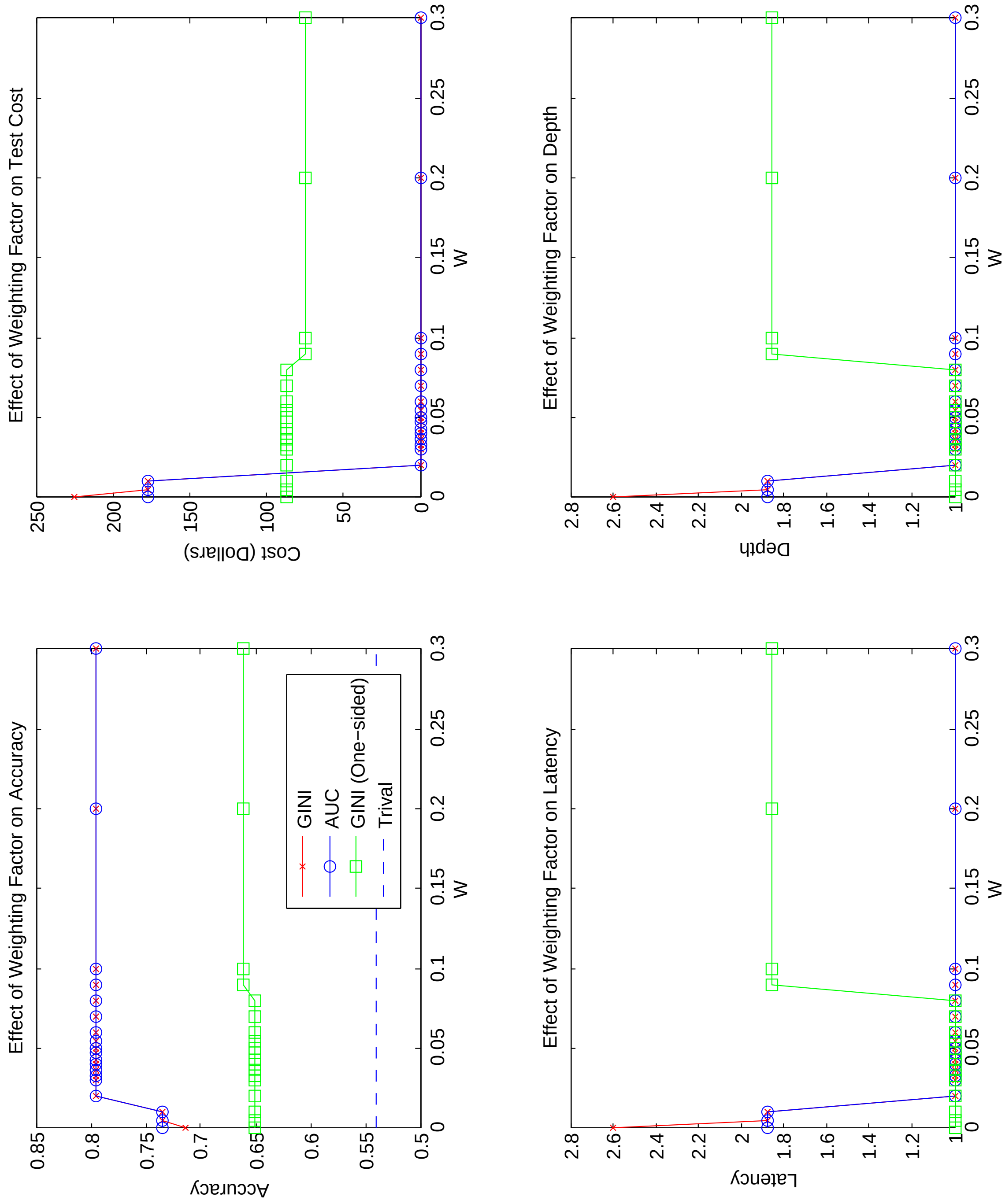


Figure 5-13: Effect of test cost weighting on classifier performance (Clev)

Chapter 6

Design for an Embedded Implementation

6.1 Program Flow

The basic flow of the embedded code is shown in algorithm 6.1. Each step is described briefly in turn, with additional discussion covered in the following section.

Algorithm 6.1: Overview of Program Flow

```
while true do
    Collect data;
    Run classifier;
    if classifier returns answer then
        Execute desired response;
    else
        Turn on necessary sensor;
    end
    Turn off unnecessary sensors;
    Sleep until next cycle;
end
```

This loop is repeated as long as the system is active, based on the assumption that the main purpose of the system is to determine the current state and respond to it. Additional operations may be added before the collection of the data or before the transition to sleep at the end of the cycle. Note that the length of the cycle constrains the complexity of the other tasks (see section 6.2.2 for details).

Data collection is determined by the current activity level of the system, 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[1]. 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. The specifics about the timing and constraints of sensor activation are described in section 6.2.2.

Given the sensor data, it is now possible to attempt to determine the current system state. The evaluation of the classification 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 (tree node) or returning the state (tree leaf). If the returned state has a response associated with it by the application designer, it is executed. The tree itself is stored in a linear array constructed from the MATLAB tree structure. The conversion process is described in section 6.2.1, including the handling of passive triggers and determination of the classifier update rate.

The above procedure can fail when the data necessary to determine the feature are not available, either because the sensor is currently inactive or has not been active long enough to the fill the data window (given that this is a tiered wakeup system, this will not be a rare occurrence). In this circumstance, the only solution is to add the part to the list of active sensors (if it hasn't been already) 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

[1]If some sensors are sampled at a lower rate than others, the system will not activate the less frequently used devices during some cycles.

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 section 6.2.3 (tree output smoothing is also covered).

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 and the associated switches.

6.2 Specific Issues

6.2.1 Conversion from MATLAB

The process of converting the tree structure from MATLAB into embedded code is mostly transcription. An array of data structures is created, with each index representing a node in the tree. The data structure contains:

- Index of the sensor tested at this node (0 for leaf)
- Index of sampling rate (of above sensor)
- Index of feature calculated (with the above sensor and rate)
- Threshold for comparison
- Index of left child (feature less than threshold)
- Index of right child (greater than or equal)
- Node state assignment (only relevant for leaves)
- List of sensors in use at this node
- Maximum sampling rate of sensors in use at this node
- Order of activation of sensors (see section 6.2.2)

The tree structure in MATLAB is compact, so there are no empty indices in the array. The first three values in the data structure are all extracted from a single index used in the classifier training process, which treats each <sensor, sampling rate, feature> triplet as a separate vector of values. These parameters can no longer be treated as distinct when the same sensor samples are used to calculate multiple features, possibly at multiple sampling rates. Similarly, while the training algorithm needn't keep track of the sensors in use at each node, it is fundamental that the embedded implementation do so. Each node contains such a list, which can be constructed with a simple traversal of the tree, allowing the algorithm to determine which previously active sensors are no longer in use when the evaluation of the tree transitions to a higher (nearer to the root) node. The rest of the values for the data structure are copied directly from MATLAB.

The update rate of the decision tree must also be considered, as it is not fixed by the classifier. The maximum useful rate is the highest sensor sampling rate used in the tree and the lowest reasonable rate is the step size of the features created for the classifier - one quarter of the window size. The choice must balance the difficulties of having the update rate change as the activity level changes and the desire to take advantage of and/or respond to those changes. Using the current maximum rate allows the system to use all the data it collects since the tree would update each cycle. However, the variation in update rate complicates the de/activation of sensors and any output smoothing which may be used. By contrast, a fixed update rate (mostly likely the lowest rate in the tree) reduces power usage and makes processes related to updating the tree (*e.g.* output smoothing and the response function) more consistent. However, this may make the system slower to respond in certain situations. For systems with less than an order of magnitude between the highest and lowest rate (such as our example system), the latter is probably the best choice. For systems which a greater spread or strict latency demands, the former would be preferred. Note that the sampling rate of the sensors is unchanged by this choice.

MATLAB also provides direct function calls for the four windowed features used in this implementation of the framework: maximum, minimum, mean and variance. Recursive algorithms for all four are presented in algorithm 6.2 through 6.5. The first two are straight-forward. For mean, we use the fact that:

$$\hat{\mu} = \frac{\sum_{i=1}^{n} x}{n}$$
$$\Rightarrow \hat{\mu} \propto \sum_{i=1}^{n} x \equiv f_\mu$$

where n is the number of points in the window. While differing from the mean by a multiplicative factor, the value f_μ preserves the ordering. This value is then used in the formula for variance, which is based on:

$$\hat{\sigma^2} = \frac{\sum_{i=1}^{n} x^2}{n-1} - \frac{(\sum_{i=1}^{n} x)^2}{n(n-1)}$$
$$\Rightarrow \hat{\sigma^2} \propto n(\sum_{i=1}^{n} x^2) - (\sum_{i=1}^{n} x)^2$$
$$= n(\sum_{i=1}^{n} x^2) - (f_\mu)^2 \equiv f_\sigma$$

Again, we see that f_σ is proportional to the variance and preserves the ordering. To calculate these values efficiently, we introduce two new data arrays: Σ_i, the running sum of all the data from 1 to i, and Σ_i^2, the running sum of the square of the data from 1 to i. Whenever a sensor is active, all three data arrays are kept up to date for each frequency in use. While this requires a larger memory footprint and a slight increase in power usage, it avoids a large amount of latency and bookkeeping issues. Further, on a tree-by-tree basis, it is possible to reduce the complexity of this approach by not filling arrays for features which are *never* used in the tree[2].

Algorithm 6.2: Recursive Windowed Mean(x_{n+1})

```
  Data: $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$          /* Running sums for last n points */
  Input: $x_{n+1}$                                       /* Newest point */
  Output: Mean of $x_2, x_3, \ldots, x_{n+1}$
1 $\Sigma_{n+1} = \Sigma_n + x_{n+1}$;
2 return $\Sigma_{n+1} - \Sigma_1$
```

[2]Calculating the variance feature requires the mean information, even if it is not used directly.

Algorithm 6.3: Recursive Windowed Variance(x_{n+1})

```
Data: $\Sigma_1, \Sigma_2, \ldots, \Sigma_{n+1}$          /* Running sums for last n+1 points */
Data: $\Sigma_1^2, \Sigma_2^2, \ldots, \Sigma_n^2$        /* Running sum square for last n points */
Input: $x_{n+1}$                                           /* Newest point */
Output: Variance of $x_2, x_3, \ldots, x_{n+1}$
$\Sigma_{n+1}^2 = \Sigma_n^2 + (x_{n+1})^2$;
return $n(\Sigma_{n+1}^2 - \Sigma_1^2) - (\Sigma_{n+1} - \Sigma_1)^2$
```

Algorithm 6.4: Recursive Windowed Maximum(x_{n+1})

```
Data: $x_1, x_2, \ldots, x_n$                                /* Previous n points */
Data: currMax                                                /* Maximum among those points */
Input: $x_{n+1}$                                             /* Newest point */
Output: Maximum of $x_2, x_3, \ldots, x_{n+1}$
if $x_1 =$ currMax then                                      /* Maximum fell off of end */
    currMax $= 0$;
    for $i \leftarrow 1$ to $n$ do
        if $x_i >$ currMax then currMax $= x_i$;             /* Find new maximum */
    end
else
    if $x_{n+1} >$ currMax then currMax $= x_{n+1}$;         /* New maximum?  */
end
return currMax
```

Algorithm 6.5: Recursive Windowed Minimum(x_{n+1})

```
Data: $x_1, x_2, \ldots, x_n$                                /* Previous n points */
Data: currMin                                                /* Minimum among those points */
Input: $x_{n+1}$                                             /* Newest point */
Output: Minumum of $x_2, x_3, \ldots, x_{n+1}$
if $x_1 =$ currMin then                                      /* Minimum fell off of end */
    currMin $= 0$;
    for $i \leftarrow 1$ to $n$ do
        if $x_i <$ currMin then currMin $= x_i$;             /* Find new minimum */
    end
else
    if $x_{n+1} <$ currMin then currMin $= x_{n+1}$;         /* New minimum?  */
end
return currMin
```

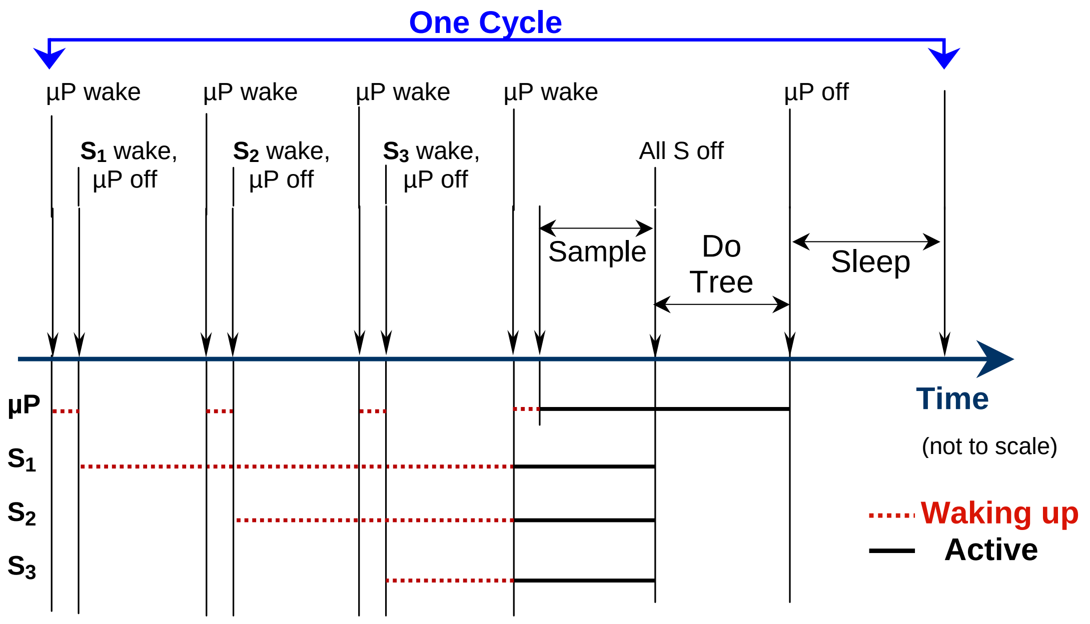


Figure 6-1: Timeline of a single cycle of the embedded hardware.

6.2.2 Power cycling

While this framework primarily saves power by not turning on unnecessary sensors, there are also significant savings to be had by power cycling both the sensors and processor. The schedule which we will use is outlined in figure 6-1, where dashed lines beside a part name indicate that they are waking up, while solid lines indicate that they are fully active. There are three distinct phases. During the first, the processor intermittently wakes up to power on the individual sensors in use in the given cycle. The activation of the individual sensors is done in decreasing order of their wake up time, such that they all become active at the same time. This allows them to be sampled at roughly the same time, minimizing the clock skew in the data. In the second, each of the (now active) sensors are sampled and the data is stored. Currently, all the sensors are deactivated at once, though they may be shut off immediately after they are sampled if desired. However, the two order of magnitude difference between the ADC sampling period and the sensor wake up period makes this a minor improvement. Finally, the tree classifier itself is executed as detailed previously.

The energy used by this scheme is:

$$\begin{aligned}E_{\text{Cyc}} = (n+1)(E_{\mu\text{P},\textsc{Wake}}) + \sum_i^n E_{S_i,\textsc{Wake}} \\ + n(E_{\mu\text{P},\textsc{Adc}}) + n(t_{\mu\text{P},\textsc{Adc}}) \sum_i^n P_{S_i,\textsc{On}} + E_{\mu\text{P},\textsc{Tree}} + E_{\mu\text{P},\textsc{Resp}}\end{aligned} \quad (6.1)$$

where n is the number of active sensors in this cycle. This formula (and therefore, the scheme) requires that:

$$t_{\text{Cyc}} > 2t_{\mu\text{P},\textsc{Wake}} + \max_i(t_{S_i,\textsc{Wake}}) + n(t_{\mu\text{P},\textsc{Adc}}) + t_{\mu\text{P},\textsc{Tree}} + t_{\mu\text{P},\textsc{Resp}} \quad (6.2)$$

where t_{Cyc} is the inverse of the current highest sampling rate. If equation 6.2 is violated, we proceed by selecting the sensor which takes the longest to wake up and making it continuously active. This process is repeated until the constraint is met (which could possibly require making all of the sensors continuously active[3]). Once the sensors to be power cycled have been determined, the timing is fairly straight-forward. Ordering these sensors in decreasing order of wake up time, the sensors are activated at:

$$t_i = t_{S_1,\textsc{Wake}} - t_{S_i,\textsc{Wake}}$$

where t_i is the activation time for S_i, with $t = 0$ being the time when the processor first becomes active. Note that $t_1 = 0$ and that $t_i \geq 0 \forall i$ by construction.

Execution of this scheme in hardware is straight-forward. The values t_i can be precomputed and stored in the tree, with a special value set for those sensors which are continuously active[4]. The times are loaded into a bank of timer compare registers, the counter is started and the processor is put to sleep. The compare module generates a series of interrupts which wake the processor at the predetermined times to activate the appropriate sensors. Note that the reduced accuracy of the clock while the processor is asleep is a point of concern. For the most common case, a 32.768 kHz watch crystal, the period is 30.5 μs.

[3]If this occurs often, it may be necessary to rethink the choice of sensors for the application. If the constraint cannot be met at all, the design flaw is fundamental.

[4]If the sampling rate of the active sensors differ, there will need to be one set of t_i for each permutation of active sensors within a cycle.

For most sensors, it is easiest to convert the value of t_i into clock cycles and simply round down (which increases the wake up time). For high power sensors[5], where the extra idle time consumes significant power, it may be more efficient to round up the value of t_i and busy-wait the processor instead (specifically, if $P_{S,\mathrm{ON}} t > P_{\mu\mathrm{P},\mathrm{ON}}(30.5\,\mu\mathrm{s} - t)$, where t is the extra idle time). This savings, however, should be considered relative to the total power usage of the device (both during activation and use) and the added complexity introduced into the system.

Finally, it should be mentioned that all reasonable timing schemes use roughly the same amount of power, differing only in their length and the amount of clock skew. The simplest scheme is simply to wake up and sample each of the sensors consecutively. This requires a minimum of bookkeeping, but take substantially longer ($\sum_i t_{S_i,\mathrm{WAKE}}$ rather than $\max_i(t_{S_i,\mathrm{WAKE}})$). It is also possible to simply run the current scheme in reverse, activating all of the sensors simultaneously and then sampling each when it becomes active. This increases the skew, though it will slightly lower the peak power draw of the system (assuming that power usage increases with time during wakeup).

6.2.3 Noise and Timing Issues

The final implementation issue considered is that of noise, specifically with respect to sensor activation and deactivation. 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/n'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 quite a large latency while the delay caused by waiting for confirmation

[5]Roughly defined as those for which $P_{S_i,\mathrm{ON}} \gg P_{\mu\mathrm{P},\mathrm{ON}}$

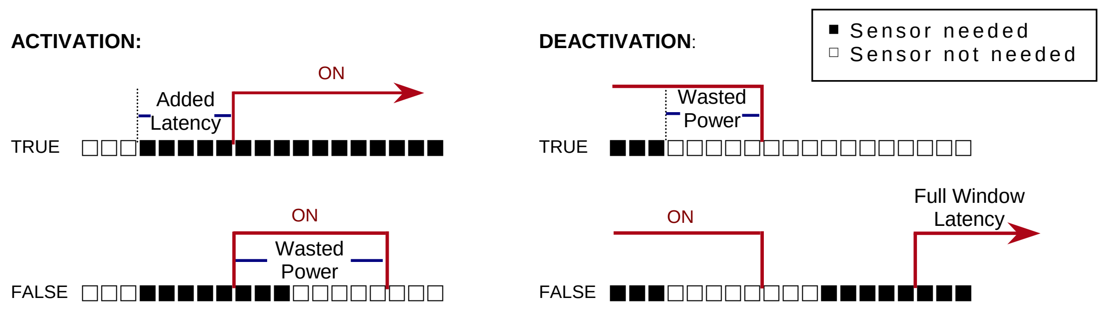


Figure 6-2: Latency and wasted power for various sensor de/activation scenarios

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 6-2, 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 twentieth of the window size. The off latency, by contrast, will be designed to be proportional to the amount of data potentially lost to take into account both fully active sensors and those which are turning on (but have not filled the data window). Values should be on the order of a quarter to half of the amount of stored data.

Depending on the choices for the de/activation latencies, some smoothing of the output state may also be desired. As with the sensors, the goal is 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 tenth to one half of the window size.

Chapter 7

Analysis

7.1 Testing Scenario

As a detailed testing scenario for this framework, we decided to mimic the wearable gait laboratory detailed in section 1.3.1. We decided on this application because a similar system was originally built using The Stack, it has a wide range of possible uses and we are familiar with the application.

The hardware setup used was a reduced set of that in [79]. An embedded module was built using the master board, the IMU board[1] and the storage board (as well as the same power regulation board). The tactile board (and insole) and sonar were omitted. The stack was then attached to the heel of the user's shoe with the same attachment as shown in figure 4-2.

Our goal was to collect a data stream which contained a 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)[40]. This data set allows us to create classifiers which attempt to separate

[1] The measurement range (for both acceleration and angular velocity) for this board exceeds the measured extrema of human ambulatory motion reported in [126].

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 for treating PD patients, a doctor would be most interested in collecting information about the frequency and parameters describing the patient's shuffling episodes[131]. For patients with total knee replacement, activities such as ascending stairs (where the knee flexion is $> 90°$) are the most important to measure[60]. Both of these cases allow for far more complex and richer classifiers than those used to simply separate ambulatory from non-ambulatory (roughly: still) states.

Data was collected using the two step process described in section 5.1.1. The active data streams contained two separate segments of at least 2 minutes in length of each of the non-walking motions. These motions were collected individually and in isolation, since only the states (and not the transitions between them) were of interest. Two segments each of fast and slow gait were captured in a similar fashion. This data was collected in a single session lasting approximately one hour and was recorded with a video camera. Annotations were later added to the data stream based on the camera's time stamp.

Several hours after the active data stream was taken, a further data stream was collected for the purposes of simulating the operation of the classifier in real-time. In the course of a single session, the wearer performed two segments (roughly 10-15 seconds each) of each of the motions of interest, with segments of level gait in between (roughly 5 seconds). The total running time of this stream is 200 seconds. Note that the data from this stream was not used to either grow, prune or test the classifier to allow for an independent real-time simulation. Details of the collection of this and the above stream can be found in appendix B.

The long-term data stream was designed to collect data representing the everyday activities of an office-bound worker (*i.e.* the author). Two hours of data were collected with the subject sitting at his desk performing a number of basic activities – typing, reading, searching for papers, *etc.* Non-ambulatory motions such as adjusting the position of the feet, moving from one desk to another (while still in a desk chair) and waving the feet under the desk were collected. The only high energy states collected were two segments of walking to and from a kitchen area. Note that this also included standing up and sitting down, two activities not considered above. The subject recorded his activities in a simple diary. Passive activities were marked at 15 minute intervals and the time (to the minute) was recorded when interesting motions took place. Annotations were later added to the data stream using the diary as a rough guide, together with visual inspection to more precisely mark the times.

While data was collected for only a single individual, studies suggest that intersubject variability of heel motion is minimal[125]. However, it should be noted that variations in gait for a user from day to day can be quite large, and more data streams would be necessary for a more general classifier. This is of little concern, as the goal is to test the capabilities and flexibility of our proposed classifier construction technique, not to solve the problem of wearable gait recognition.

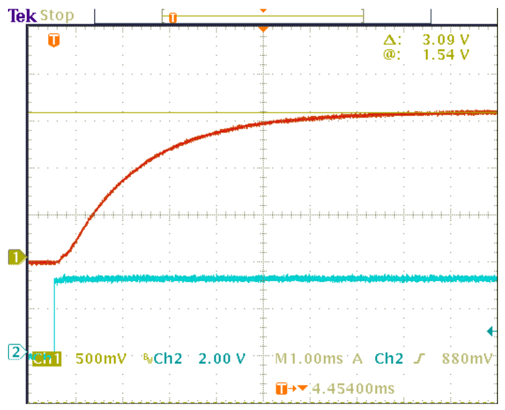
Finally, note that the data was collected at 200 Hz (the chosen rate for human activity from section 5.1.1). Versions of the data streams for sampling rates of 25, 50 and 100 Hz were constructed in MATLAB using the `downsample` function, which does not apply an anti-aliasing filter.

The specific parameters of these data sets are examined in the next section. They are then used to train a number of different classifiers and simulate the real-time operation of those classifiers.

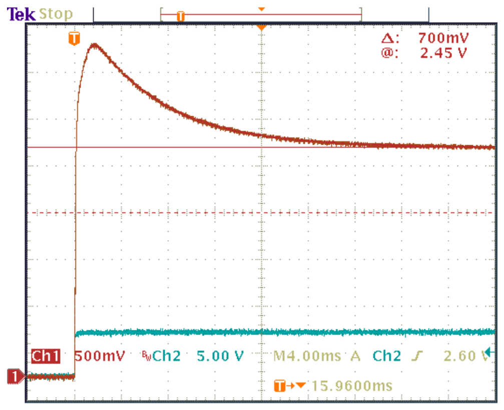
7.2 Parameters of the Training Set

7.2.1 Sensors

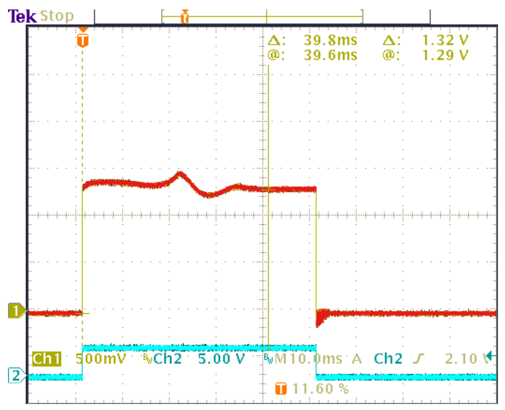
To determine the test cost of the sensors while power cycling, it is necessary to know the wakeup time of the device or the settling time of the output. It should be noted that the availability of this information is spotty at best - provided on some data sheets while completely ignored on others. Further, 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



(a) ADXL202

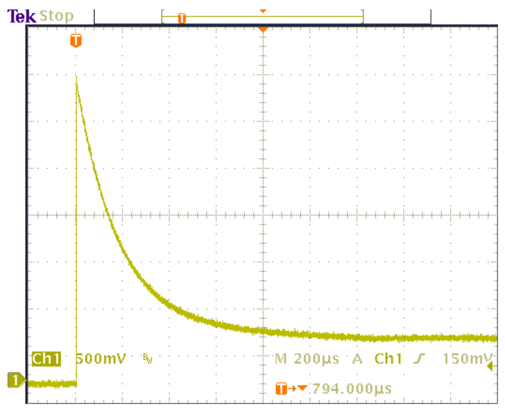


(b) ADXRS300

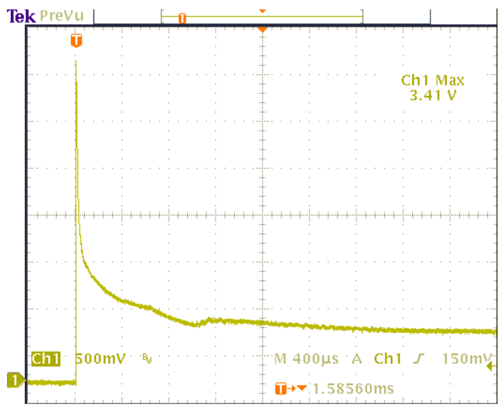


(c) ENC03J

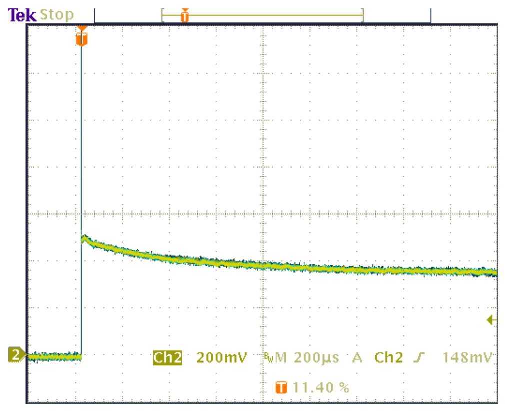
Figure 7-1: Oscilloscope captures of sensor settling times



(a) ADXL202



(b) ADXRS300



(c) ENC03J

Figure 7-2: Oscilloscope captures of sensor wake up current

Sensor	E_{WAKE} (μJ)	t_{WAKE} (ms)	P_{WAKE} (mW)	P_{ON} (mW)	Effective t_{WAKE} (ms)
ADXL202	13.5	8	1.7	1.5	9.1
ADXRS300	890	35	25.5	24.8	36
ENC03J	478	40	12	11.9	40.2

Table 7.1: Sensor wake up and active power usage

sensors (*e.g.* those that need to charge internal capacitor or equilibrate filters), it may well be noticeably more. For the sensors used in the test set, settling time and power draw during that period was measured directly.

Figure 7-1 shows the settling time for the sensors used. For the ADXL202 and the ADXRS300, the settling time of the output value (to the zero bias offset) was used. The values found were consistent with those listed on the data sheets (see table 4.1). For the ENC03J, the output only becomes valid after the (somewhat) sinusoidal signal in the output. The point where the data becomes valid is denoted by the horizontal line on the graph and was found experimentally to be roughly 40 ms.

Figure 7-2 shows the current drawn during wake up by the various parts. These values where collected using a high-side sense resistor of 100 Ω for the ADXRS300 and the ENC03J and 1000 Ω for the ADXL202. With the exception of the ADXL202, the wake up power was not significantly different than the power used by the part when active.

Table 7.1 gives the parameters calculated from these measurements. The wake up energy is simply the integral of the current drawn during wake up multiplied by the known supply voltage. The wake up time is taken directly from the scope captures, allowing the wake up power to the calculated. The power drawn when the sensor is active is calculated based on the quiescent current draw (again found in the scope captures). Finally, the effective wake up time is the duration for which the power usage of leaving the sensor active is equivalent to that of power-cycling it. It is this value, rather than the wake up time, which should be used to determine if power-cycling is beneficial. It is only significantly different in the case of the ADXL202.

The energy necessary to collect sensor data using the ADC is also considered here. This value depends on two parameters: the energy used to sample the sensor value and the energy

Feature	# Add	# Multiply	# Compare	E_{FEAT} (nJ)
Maximum	0	0	2	24
Minimum	0	0	2	24
Mean	2	0	0	18
Variance	4	3	0	144

Table 7.2: Number of operations per time step to calculate various features

used in the analog to digital conversion itself. The former depends on the output resistance of the sensor's signal processing chain. Assuming that the final component in the chain is an operational amplifier, this value will be roughly zero. Thus, a sample time of $1\,\mu$s is required. Combined with a conversion time of $3\,\mu$s and a power usage of 2.5 mW [118], we find $E_{\text{ADC}} = 10\,\text{nJ}$.

Collecting data from the tilt switches is a different matter because of the binary nature of the data. The specific collection procedure requires two port writes (to drive pins high) and two port reads (to determine which switches are active). Using operational data of the MSP430F1610 given below, the data collection is found to require 24 nJ. Since four binary values are collected from the four-way tilt switch, the energy consumed per read is less than that used by the ADC, though not substantially.

7.2.2 Features

While the test cost of the features will vary with the processor chosen, a general analysis based on the number of operations is still possible. Table 7.2 lists the number of add, multiply and compare operations necessary at each time step for the four features used on this work.

For the specific case of the MSP430F1610[118], each instruction cycle draws 1.5 nJ. Add operations were found to take 6 cycles, compares take 8 cycles and multiplication requires 24 cycles. These values assume that the operands are stored in local RAM. If the data are stored in registers or external memory, the operations will take fewer or more cycles, respectively. Further, compiler optimizations[2] are not considered here. Using the value above, the energy used for each feature was calculated and added to table 7.2.

[2]These include shadowing commonly used data in registers and exploiting the "free" addition operation of the multiply-and-accumulate functionality of the hardware multiplier.

Sensor	E_{WAKE} (μJ)	E_{FEAT} (μJ)	E_{ADC} (μJ)	Total (μJ)	E_{WAKE} dominates?
ADXL202	13.5	0.210	0.010	13.72	Y
ADXRS300	890	0.210	0.010	890.22	Y
ENC03J	478	0.210	0.010	478.22	Y
Tilt	0[a]	0.210	0.024[b]	0.234	N

[a]Passive device. No wake up time or static power draw

[b]Energy for digital sampling

Table 7.3: Total energy usage of the sensors

	Power usage[a] (μW) at			
	25 Hz	50 Hz	100 Hz	200 Hz
ADXL202	371	711	1391	1530
ADXRS300	22343	24831	24831	24831
ENC03J	11931	11931	11931	11931
Tilt	5.85	11.7	23.4	46.8

[a]Gray cells indicate that the sensor cannot be power-cycled at this rate.

Table 7.4: Power usage of sensors by sampling rate.

7.2.3 Test Cost

Given the above discussion, it is possible to calculate the total test cost for each of the features of each of the sensors. In this case, we will actually calculate the energy usage to calculate all of the features for each sensor. By computing all of the features whenever a sensor is in use, it is possible to avoid the increased latency from having to collect a full window worth of data whenever a new feature is requested by the tree (see section 6.2.3). As will be seen, this choice has a negligible effect on energy usage with the exception of the tilt switch. However, this is only a minor concern as there are two orders of magnitude difference between the test cost of the tilt switch and the next cheapest sensor. Table 7.3 shows the total energy use of each sensor per sample.

Table 7.4 gives the power usage of each sensor (with all features calculated) at various sampling rates. Note that the ADXRS300 cannot be duty-cycled past 33 Hz, the accelerometer cannot be duty-cycled past 125 Hz and the ENC03J cannot be duty-cycled at all. In these cases, it is P_{ON} (rather than P_{WAKE}) which is used in the calculations.

Activity	Examples
Level Gait	305
Uphill	229
Downhill	264
Ascend Stairs	222
Descend Stairs	221
Shuffling Gait	482

Table 7.5: Number of examples of each activity in training set

7.2.4 Priors and Number of Examples

Because of wide variability in the amount of ambulatory activity between individuals, it is difficult to impossible to select a general set of priors for the motions which make up this application. However, in this specific case, it is known that classifiers constructed with the training set will be run on a known set (specifically, the continuous stream for use in the real-time simulation). Therefore, the distribution of that set is used for the priors, giving a distribution which is approximately equal for all motions. Table 7.5 gives the number of examples for each activity in the training set. Given the limited variability within the motions, this set should be adequate for a holdout training and testing procedure.

7.3 Performance of Classifier

7.3.1 Methodology

Classifiers were constructed 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. Only the active data set was used to train the classifiers. 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 (since both the classification tree and the SVM can be awoken by a simple trigger). To determine how the classifier responds to the availability of features calculated at different sampling rates ($25, 50, 100$ and $200\,\mathrm{Hz}$), the following combinations were tested:

- All the frequencies together
- Each frequency individually
- The combination of pairwise adjoining rates

The classification trees are constructed using a generalized version of the holdout procedure described in section 5.4.1. The full data set is divided into 5 segments, with one acting as the holdout test set and the other four used as the training set. The portion used for the test set will rotate, such that all five segments are eventually used. The 20% holdout pruning set (from the four segments which make up the training set) is treated in the same fashion. Overall, 25 classifiers will be built in each case. As before, accuracy and test cost will be found using the test set.

Support vector machines are trained[3] for the same tasks as above to act as a point of comparison. Both Gaussian and linear kernels are used[4], 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 will be trained for all 127[5] possible combinations of active sensors run at the maximum sampling rate.

Results from the above tasks will be plotted in the power/accuracy plane, with each classifier being a single point. An algorithm will be considered superior to another if its curve is closer to the 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. Note that it is possible (and likely) that different curves will be 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.

[3]The LIBSVM package[18] was used.

[4]For the linear kernel, values of the regularization parameter C between 0 and 1000 were tested. For the Gaussian kernel, values of C from 10 to 100,000 were tested and γ was set to 0.5.

[5]There are seven total sensors: three accelerometers, three gyroscopes and the tilt switch. The total number of combinations is $2^7 - 1$, since the case with no sensors active is uninteresting.

7.3.2 Full Sensor Set

As described above, the decision tree classifier was trained to recognize each of the six motions individually. Trees were constructed using both the AUC-based and Gini criterion with 13 different values of the weighting parameter W over the useful range of 0 to 0.29. The plots of accuracy versus power are shown in figure 7-3 and 7-4. These graphs are broken down by the subset of the sampling frequencies used in constructing the classifier. Note that the minimum of the accuracy axis is that of the trivial classifier (always negative) and the maximum of the power axis is twice the power usage of the most accurate classifier. While this leads to the truncation of some curves, the points lost were sub-optimal. The classifiers constructed using the Gini splitting criterion proved to be superior to those built with the AUC-based criteria (for this application) and therefore only the former are shown.

The most interesting feature of these graphs is the fact that training with all sampling rates available does not give the best results in any case. Rather, the tasks fall into two sets. For uphill, downhill and level gait, the 200 Hz data seems to give the best classification results in general, with either the 100 Hz, 100 Hz and 200 Hz, or 200 Hz data sets being superior at some point in the curve. The graphs of variance versus frequency show that the 100 Hz data is often only marginally noisier than the 200 Hz data, so the feature choice will depend on the minutia at each individual split and thus neither value is clearly superior. For ascending and descending stairs and the shuffling gait, the 50 Hz and 25 Hz data sets are 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.

The next step is to compare the performance of the decision tree classifiers with those obtained from using support vector machines. These results are shown in figure 7-5 and with the axes reversed in figure 7-6. In this case, the decision tree curves are the non-decreasing hull of all of the different sampling frequency combinations used in the training. 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[6].

[6]The exception is the plot for classification of shuffling gait, where the next point over for the SVM is at 1.5 mW.

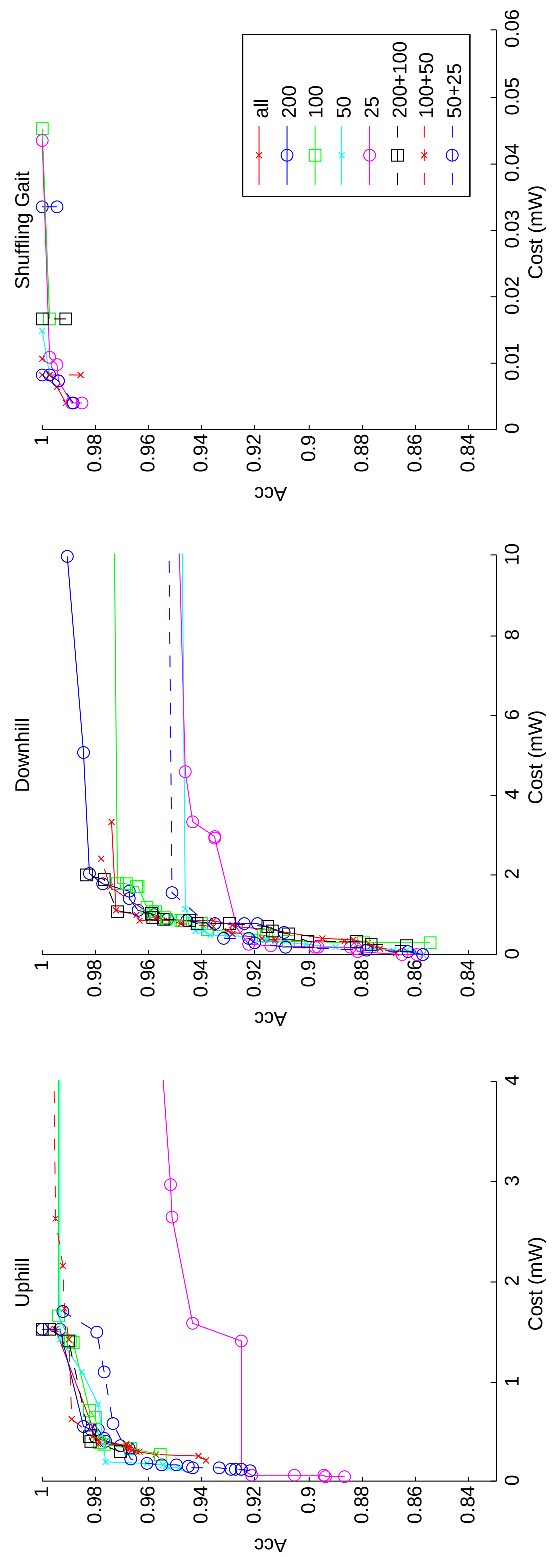


Figure 7-3: Power/accuracy tradeoff for various subsets of the data (1)

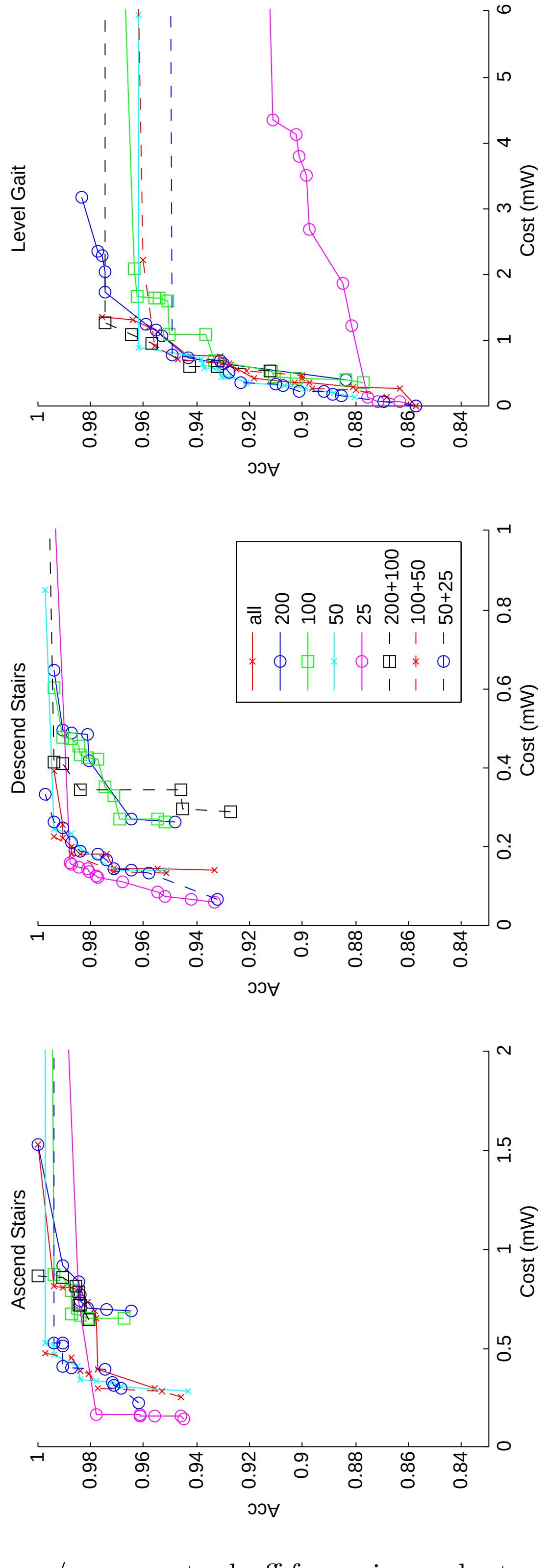


Figure 7-4: Power/accuracy tradeoff for various subsets of the data (2)

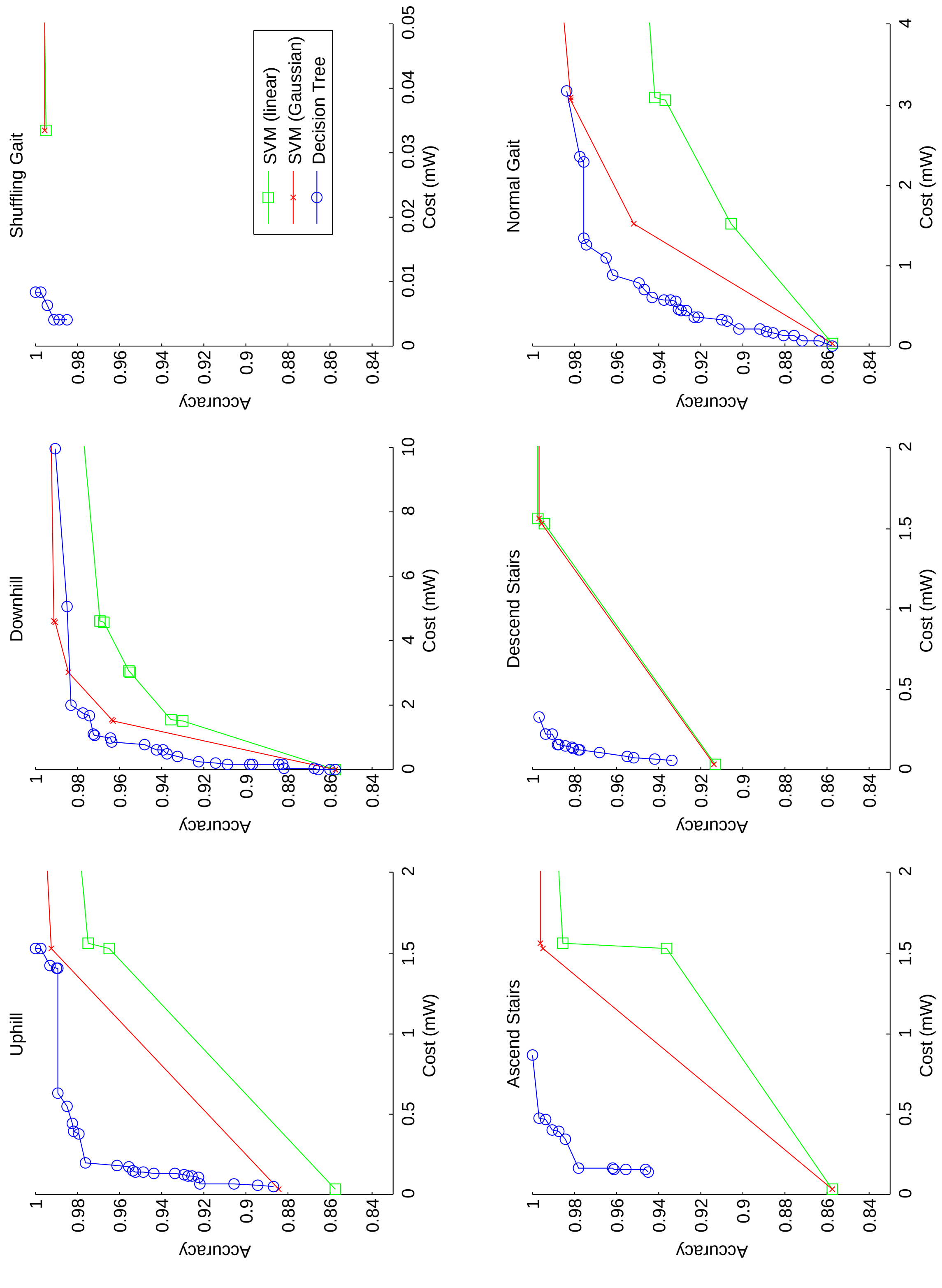


Figure 7-5: Power/accuracy tradeoff for decision trees and SVM

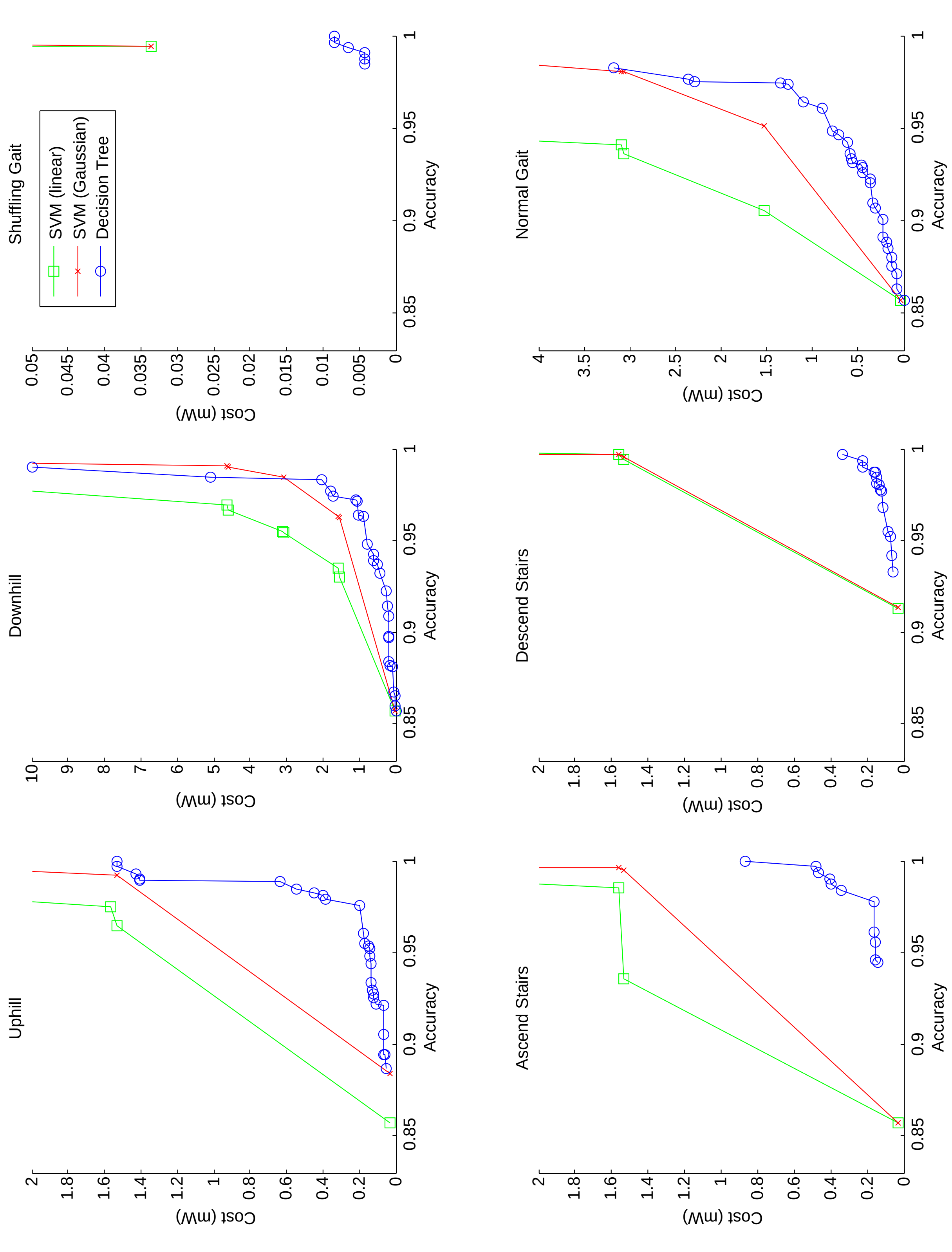


Figure 7-6: Power/accuracy tradeoff for decision trees and SVM (swapped axes)

Activity	Accuracy	Power (mW)	Sensors used
Level Gait	0.9869	4.59	3 Accelerometers
Uphill	0.9991	3.06	2 Accelerometers
Downhill	0.9901	4.59	3 Accelerometers
Ascend Stairs	0.9946	1.53	1 Accelerometer
Descend Stairs	0.9955	1.53	1 Accelerometer
Shuffling Gait	0.9952	0.0336	Tilt Switch

Table 7.6: Highest accuracy achieved by Gaussian SVM

It is noted that in all cases the decision trees over their range of operation are superior to the linear SVM, achieving the same accuracy for less power, and are superior to the Gaussian SVM in all but the downhill gait. Further, it should be noted that the Gaussian SVM is in fact 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.* $\sim$ 100 for walking). This is not only time consuming, but requires large amounts of static memory. Thus, it is important to note that this comparison is relevant numerically but impossible to be implemented practically.

That said, since the decision tree has access to all of the sensors, it is not immediately clear why it is not superior in all cases. Examining the best result for each of the SVMs provides some ideas (table 7.6).

The SVM for walking downhill uses all three accelerometers, with accuracy increasing with each one added to the classifier, indicating that a fair number of expensive sensors are necessary to achieve adequate performance. Because of the greedy training and the test cost weighting, the decision trees are unlikely to find these solutions through recursive partitioning. Even those trained without taking cost into account (the best point on the curve) do not find these solutions.

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 the more expensive sensors closer to the root node.

	Naive Decision Tree		**This Framework**		**Ratio (DT:This)**	
	Accuracy	Power (mW)	Accuracy	Power (mW)	Accuracy	Power
Level Gait	0.9774	41.35	0.9742	1.264	1.003	32.7
Uphill	0.9914	1.53	0.9845	0.548	1.007	2.79
Downhill	0.9822	29.42	0.9818	2.049	1.001	14.36
Ascend Stairs	0.9968	14.99	0.9935	0.467	1.003	32.10
Descend Stairs	0.9876	4.62	0.9876	0.1592	1.000	29.02
Shuffling Gait	1.0000	1.53	0.9912	0.0042	1.008	364.3

	Gaussian SVM		**This Framework**		**Ratio (SVM:This)**	
	Accuracy	Power (mW)	Accuracy	Power (mW)	Accuracy	Power
Level Gait	0.9869	4.59	0.9742	1.264	1.013	3.63
Uphill	0.9991	3.06	0.9845	0.548	1.015	5.58
Downhill	0.9901	4.59	0.9818	2.049	1.008	2.24
Ascend Stairs	0.9946	1.53	0.9935	0.467	1.001	3.28
Descend Stairs	0.9955	1.53	0.9876	0.1592	1.007	9.61
Shuffling Gait	0.9952	0.0336	0.9912	0.0042	1.004	8.00

Table 7.7: Comparison of best practises classifiers to this framework

This strongly supports the claim that a hierarchical activation system can provide strong classifiers for reduced cost.

To demonstrate this point, table 7.7 compares three different classification algorithm. The first two are what might be considered the current best practises used for constructing SVMs and decision trees, with the third being the framework presented here. For SVMs, feature selection algorithms are almost always used. Therefore, the accuracy and power usage given are for the best classifiers found above. For decision trees, it is often assumed that the construction of the classifier will (effectively) perform the feature selection, so the best tree for $W = 0$ is used. The power usage given assumes that all of the sensors in the tree are always active. For both SVMs and the naive decision trees, features are calculated at the maximum sampling rate - again the current standard practise (at least in the case of embedded sensors). In all three cases, 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 choosing very power hungry classifiers which are only marginally more accurate than those with more reasonable power usage.

First note that naive decision trees perform very poorly against this our framework. No classifier offers an accuracy improvement greater than 0.8%. In general, improvements in accuracy are roughly 0.3%, at the price of drawing 30 times as much power. Much of this is due to the restriction of having all sensors in the tree continuously active, though it is important to note that none of the best naive trees matches those selected by our algorithm. This suggests that growing decision trees for $W \neq 0$ has merit distinct from the assumed reduction in power, and that this small check of a pure greedy growing algorithm may have general benefits.

When comparing the SVMs, our framework still performs quite well, though the difference is not as extreme as above. SVMs offer an accuracy improvement on the order of 1%, though at a power increase of three to nine times in most cases. In this case, it appears that the hierarchical activation of sensors is what allows our framework to compete with a classifier with greater descriptive power.

7.3.3 Without Redundant Sensors

To examine the benefits of adding redundant sensing to the sensor set (in this case, tilt switches which shadow the accelerometers for near-zero cost), the same classifiers as above were trained, omitting the tilt switches. The results are shown in figure 7-7. Note that the minimum power level is now much higher, at roughly 1.5 mW for the SVM (one accelerometer at 200 Hz) and 0.4 mW for the decision tree (one accelerometer at 25 Hz).

Again, we see that the decision tree is significantly better than the linear SVM and superior to the Gaussian SVM in most cases. Interestingly, we note that the SVM also previously benefitted from the availability of the tilt sensors, as shown by the vertical connectors in figure 7-5 which indicate that a tilt sensor (of near zero cost) has been added to the classifier (stair ascent is a particularly strong example). Comparing the decision trees trained with tilt sensors to those trained without also provides some interesting results (figure 7-8).

We note that the trees constructed to classify shuffling gait and stair descent take full advantage of the tilt sensors and therefore perform far better than ones without. For both

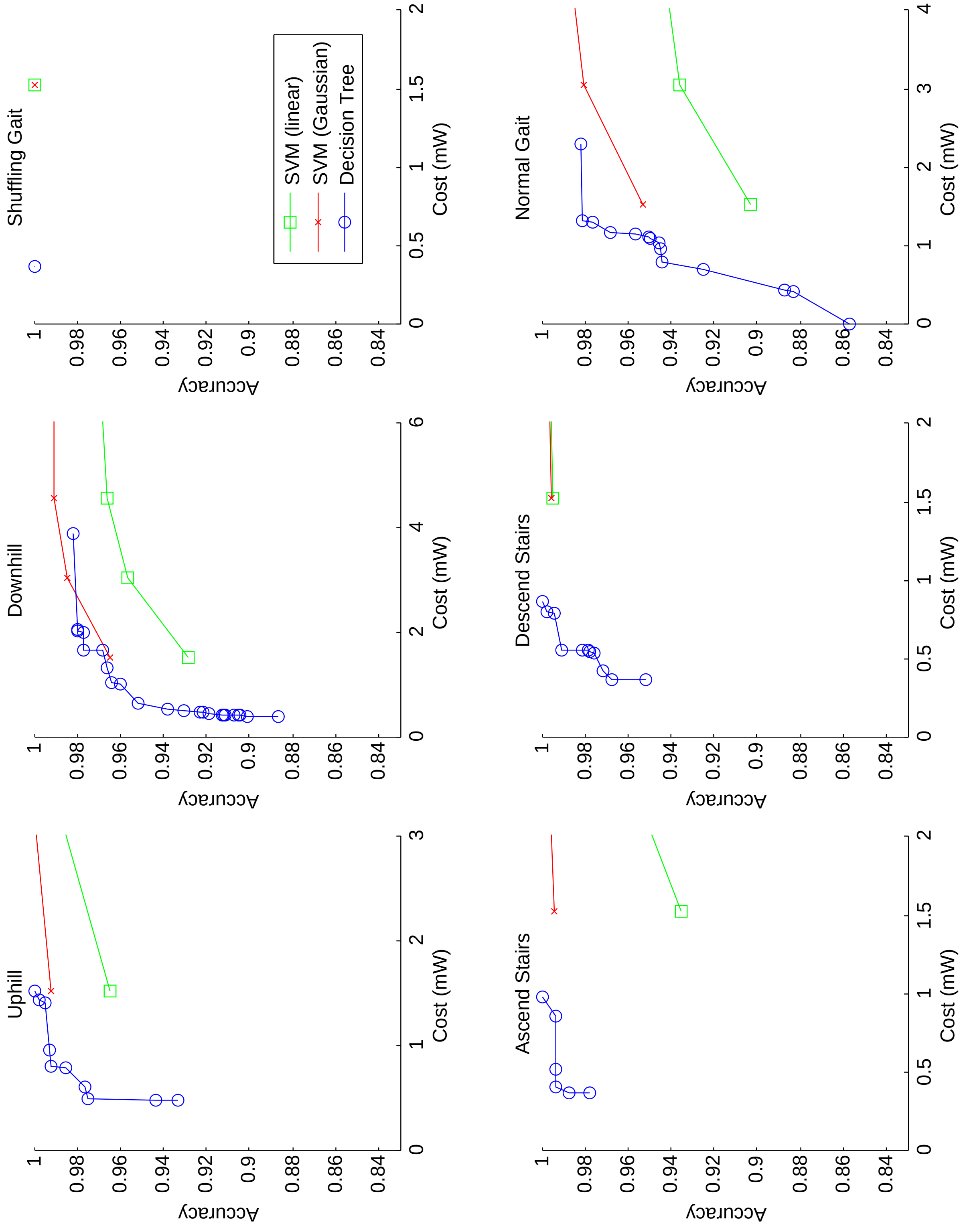


Figure 7-7: Power/accuracy tradeoff for decision trees and SVM (no tilt sensors)

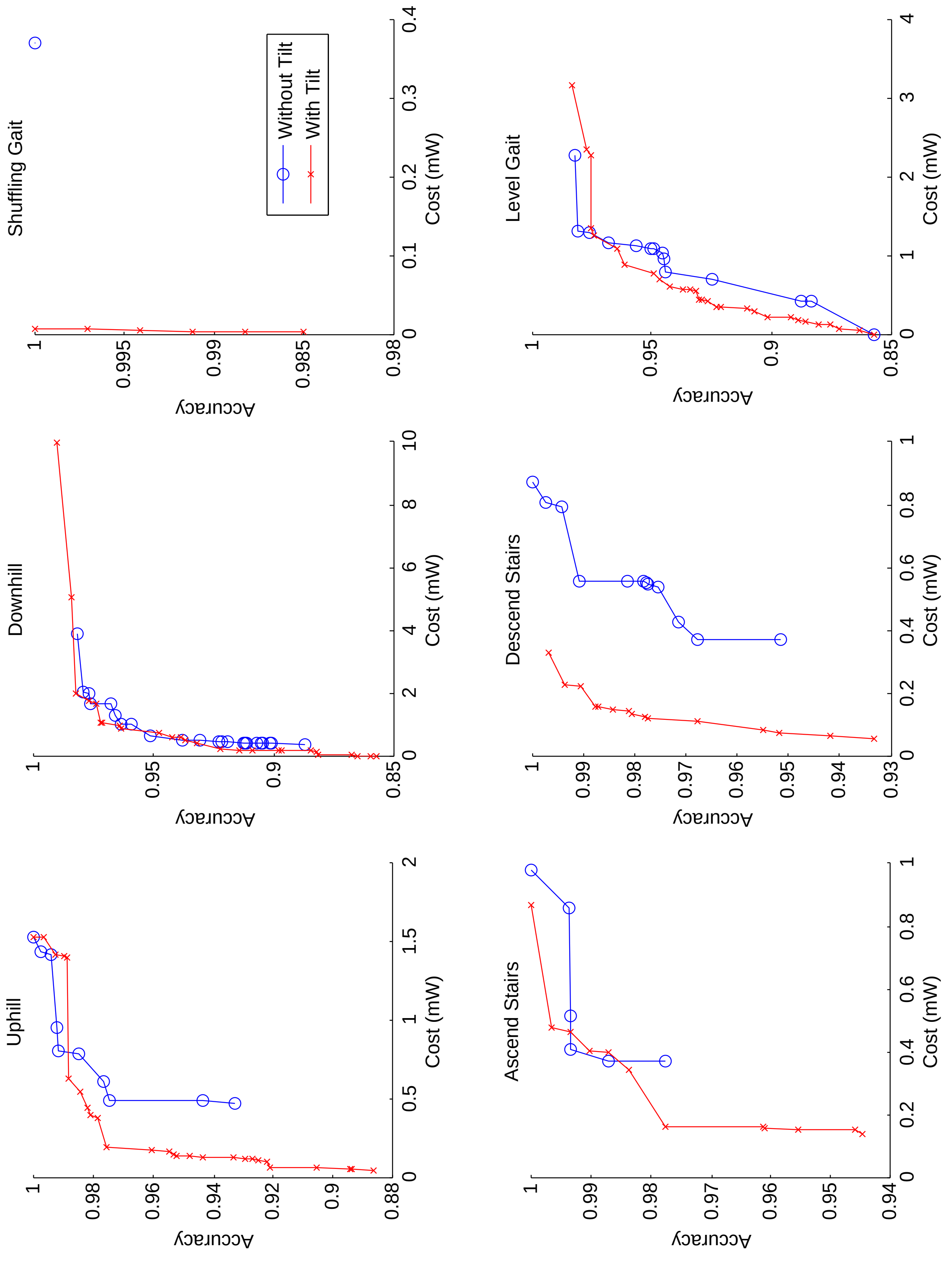


Figure 7-8: Power/accuracy tradeoff for decision trees with/out tilt sensors

uphill gait and stair ascent, the decision tree using tilt sensors dominates until the accuracy reaches 0.985, suggesting that the tilt switch is no longer useful (or used) beyond that point. Classifiers for downhill and level gait are virtually unchanged.

7.4 Simulation of Real-time Operation

Two concerns arise when using the static trees grown as part of this framework 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. These points are considered below.

Before testing can be done, representative classification trees for each of the motions must be chosen. We will use the trees chosen for table 7.7 - those at the knee of the curve. As discussed in section 7.3.2, each tree uses only a single sampling rate throughout their operation. This greatly simplifies the simulation of the real-time operation of the classifier by avoiding issues related to changes in sampling rate.

For each motion, a figure of the tree classifier itself as well as a series of plots detailing the simulated real-time operation can be found on the following pages. This simulation follows the implementation details given in chapter 6. The turn on hysteresis was set to one-twentieth of the window size, the turn off hysteresis was one-half of the window size and the output smoothing was set to one-quarter of the window size.

Each classifier was tested on the continuous stream set aside for this purpose in section 7.1. Each series of plots (*e.g.* figure 7-10) shows four different interpretations of the operation of the classifier and each is denoted with the ground truth - the hand annotation marking when the motion of interest is taking place. The first plot, with the mint green background, is the state output of the static classifier (for which all of the sensors are always active). This is a point of comparison for the real-time classifier, which cannot, by definition, respond more quickly. The second plot shows the state output for the real-time classifier. The third plot is the activation level of the classifier, defined as the number of sensors active (*i.e.* available for use) at any given point. Finally, the last plot shows the power consumption, which includes not only the sensors which are active but those which are waking up (*i.e.* for which enough data has not yet been collected for features to be calculated). 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.

We note that, in general, the state classification of the simulated real-time operation is, in fact, better than that of the static classifier. While a quantitative evaluation of false positives and negatives is not appropriate because of the conservative nature of the annotation, qualitatively the real-time classifiers have a notably lower rate of false positives with very little increase in false negatives. This is to be ascribed to the delay caused by sensor activation latency, though a higher value for output window smoothing for the static classifier might create the same effect. However, what is important is not that the static classifier could be as good as the real-time one, it is that the real-time classifier is not worse than the static one.

Table 7.8 shows the total energy used over the simulation stream for each of the classifiers. In most cases, it is less than the estimated energy use obtained by multiplying the average power usage of the tree classifier by the length of the stream. This is simply explained by the turn on hysteresis reducing sensor usage, especially in the case of short or spurious requests when the sensor is never activated at all.

Table 7.9 shows the energy saving of hierarchical activation compared to naively activating all of the sensors used in the tree. Hierarchical activation draws roughly half as much power in most cases. Even greater savings are achieved in the case of the classifiers for stair descent and level gait, where the tree rarely reaches some of the lower (and more expensive) nodes. Note that these savings are obtained on high energy data streams and the benefits would be greater in situations with lower total activity, where our classifiers would spend more time in the higher nodes.

The individual classifier responses bear additional discussion. The uphill gait classifier (figure 7-9) is unique in that it uses more energy than estimated. The explanation for this

Activity	Estimate Energy Usage (mJ)	Actual Energy Usage (mJ)	Difference (%)
Uphill	113	139	+23
Downhill	422	397	-6
Shuffling Gait	0.870	0.870	0
Ascend Stairs	96.2	79.5	-20
Descend Stairs	32.8	18.8	-70
Level Gait	257	197	-30

Table 7.8: Energy usage of trees in real-time simulation

Activity	Naive Energy Usage (mJ)	Actual Energy Usage (mJ)	Difference (%)
Uphill	322	139	-57
Downhill	945	397	-58
Shuffling Gait	0.870	0.870	0
Ascend Stairs	148	79.5	-46
Descend Stairs	154	18.8	-88
Level Gait	952	197	-79

Table 7.9: Energy savings of trees in real-time simulation

lies in the 80-100 sec segment of the stream which, as seen in the bottom two plots, has a high activation level considering that the user is apparently not walking uphill (figure 7-10). In fact, as can be seen in figure B-3, the user is actually on a shallow slope at the time. This slope is apparently steep enough to raise the activation level (and energy usage) but not enough to trigger state recognition.

The tree for downhill gait classification is quite complex (figure 7-11), which is not surprising given the apparent difficulty of the recognition task. Nonetheless, the simulated real-time classifier was able to detect roughly three-quarters of the downhill gait in the stream (see figure 7-12).

The shuffling gait classifier (figure 7-13) is, by contrast, quite simple. However, this causes a high level of false positives, since any stutter step or adjustment appears to be a shuffle (figure 7-14). The latency in the real-time operation removes a few of these, but not many. A higher level of output smoothing appears necessary for this classifier, though a more expensive classification tree using an accelerometer feature at the root may also improve the accuracy.

A few false negatives for the stair ascent classifier are caused by latency effects, but it is otherwise unremarkable. The tree is shown in figure 7-15 and the results can be seen in figure 7-16.

The stair descent classifier stands out as drawing far less power than expected in the real-time simulation. The reason for this lies in the structure of the decision tree (figure 7-17). While there are positive states at two different activation levels, only the less costly one is ever reached due to latency issues. Therefore, while the static classifier often activates three sensors, the real-time classifier only uses two (figure 7-18). Output smoothing repairs the slight defects such that they are not noticeable in the final state determination.

The real-time response of the level gait classifier (figure 7-19) appears, at first blush, to be rather poor. However, there are two important facts to note. First, even the static classifier has both high false positives and negatives (figure 7-20). Second, the segments of level gait are very short, roughly five steps or less. Therefore, not only does sensor latency make it difficult for the classifier to make the appropriate determination, but the examples of level gait themselves are closer to the transitions than to the examples in the training stream. Furthermore, the benefit of detecting such short stretches of motion is minimal at best.

Two prior points of concern can be addressed given these results. The first is the response of the classifiers to untrained states (see section 5.1.1). The simulation data stream contains such situations: one from the 85 second to 95 second mark, the other from the 140 second to the 155 second mark. In the first case, only the uphill gait classifier responds at a high activity level, which is to be expected since the untrained state closely approximates the training examples. In the second case, the shuffling gait classifier responds, though only at the beginning, which can be explained by the fact that most transitions will appear to be shuffling as the user adjusts from one motion to the next. The second is the risk of increased latency in balanced trees (see section 5.2.3). While most of the trees used in these trials are quite balanced, no unreasonably high latencies were seen, likely since sensor usage is balanced as well - *i.e.* the order of sensor activation is the same on either side of the decision tree. Thus, even if the result of the root node were to change, flipping the evaluation to the other half of the tree, all of the necessary sensors should already be active.

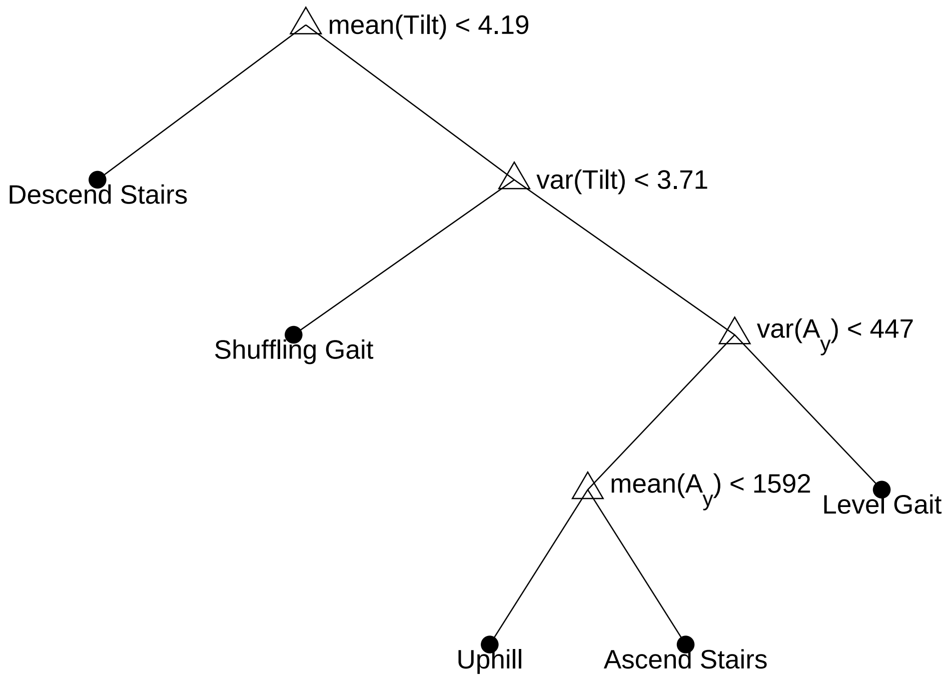


Figure 7-9: Decision tree for uphill gait classifier (f=200 Hz)

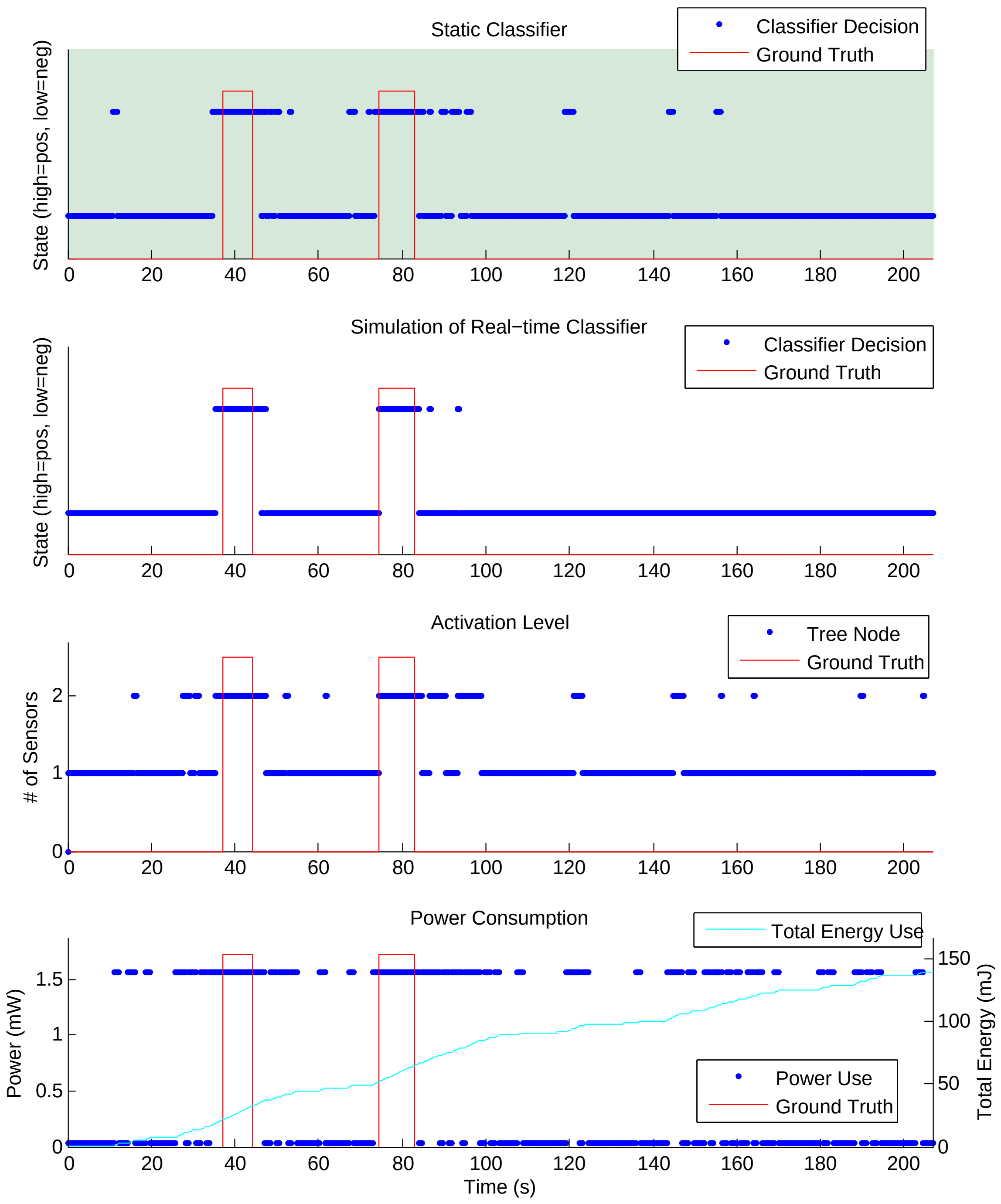


Figure 7-10: Real-time simulation of uphill gait classifier

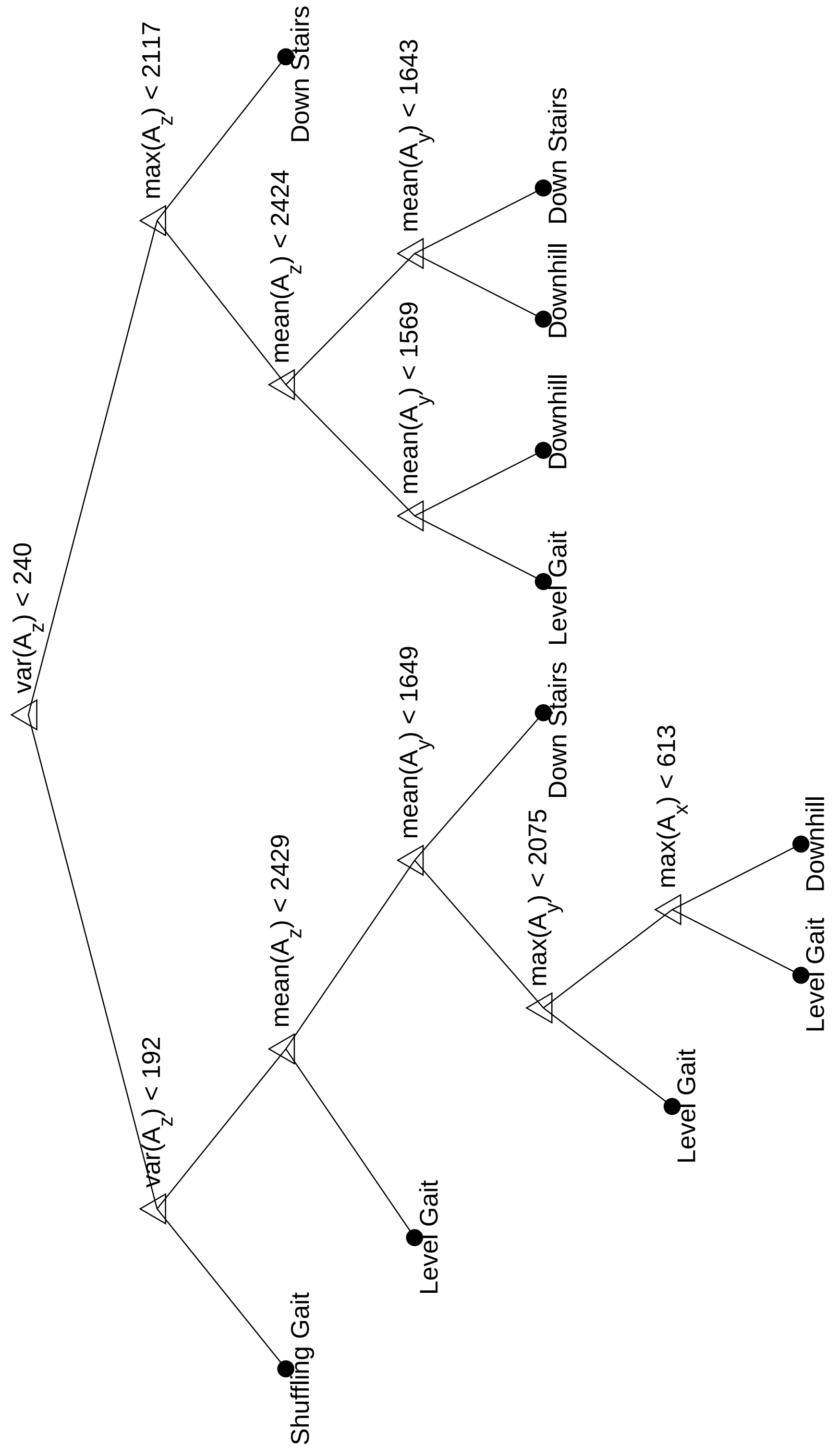


Figure 7-11: Decision tree for downhill gait classifier (f=200 Hz)

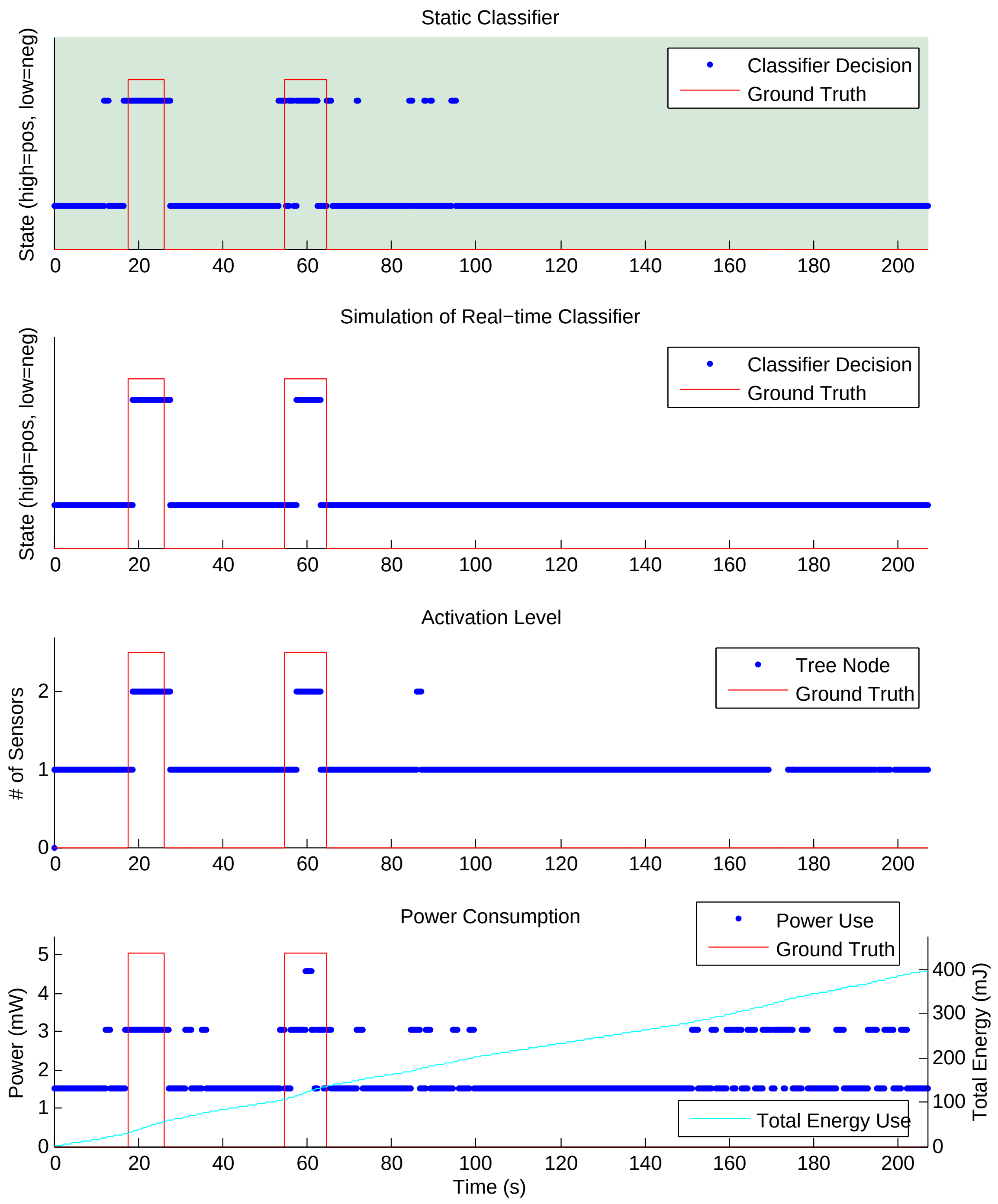


Figure 7-12: Real-time simulation of downhill gait classifier

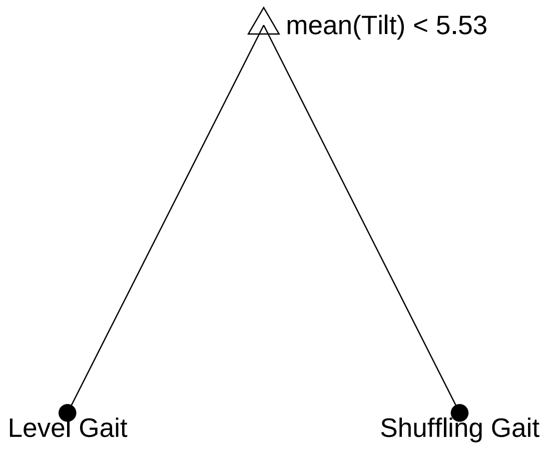


Figure 7-13: Decision tree for shuffling gait classifier (f=25 Hz)

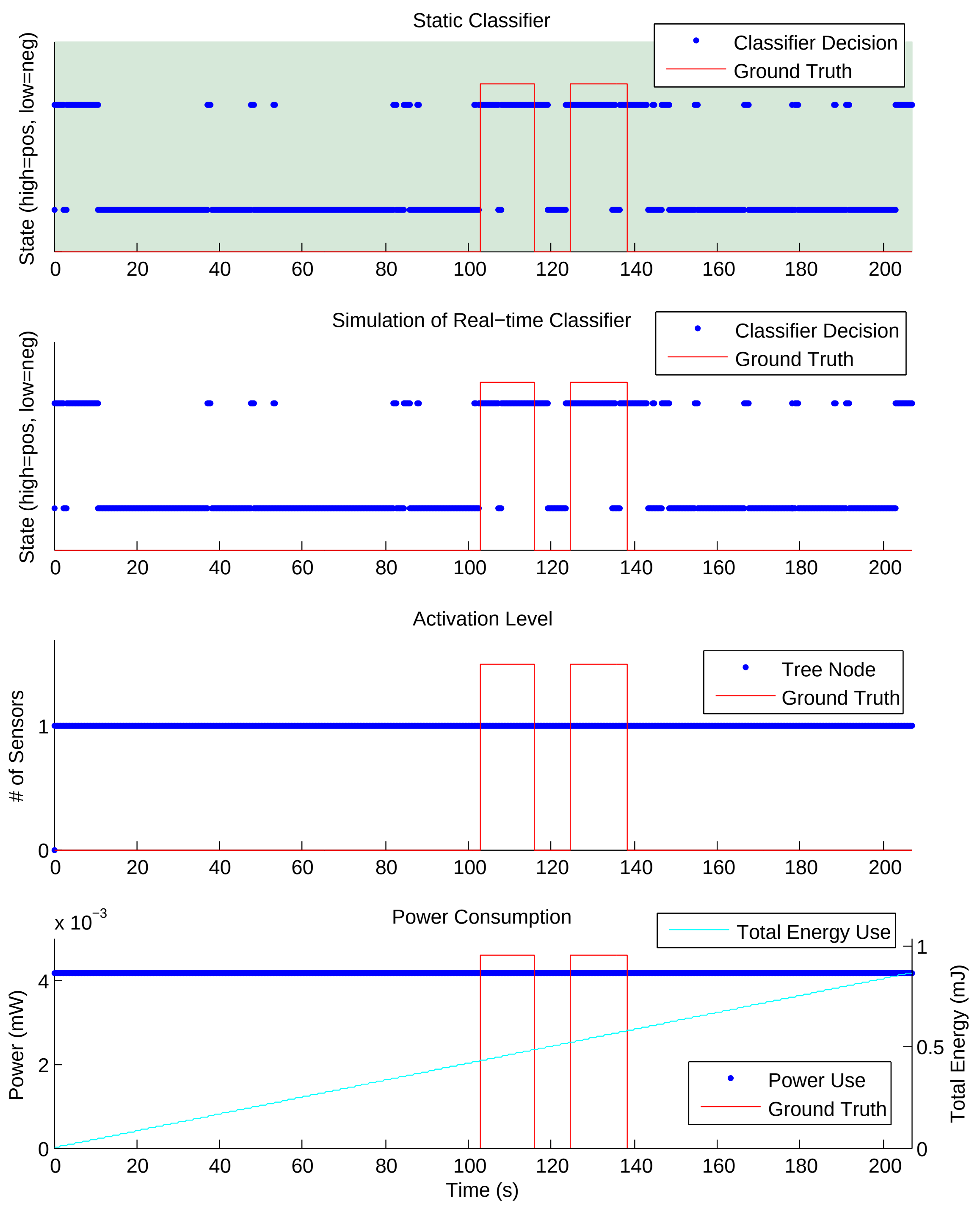


Figure 7-14: Real-time simulation of shuffling gait classifier

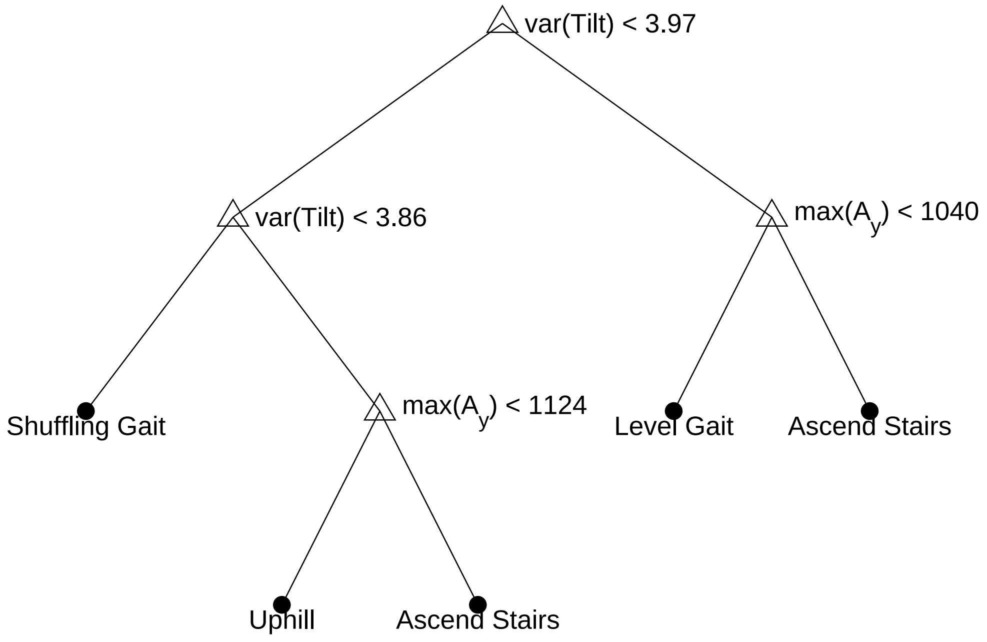


Figure 7-15: Decision tree for stair ascent classifier (f=50 Hz)

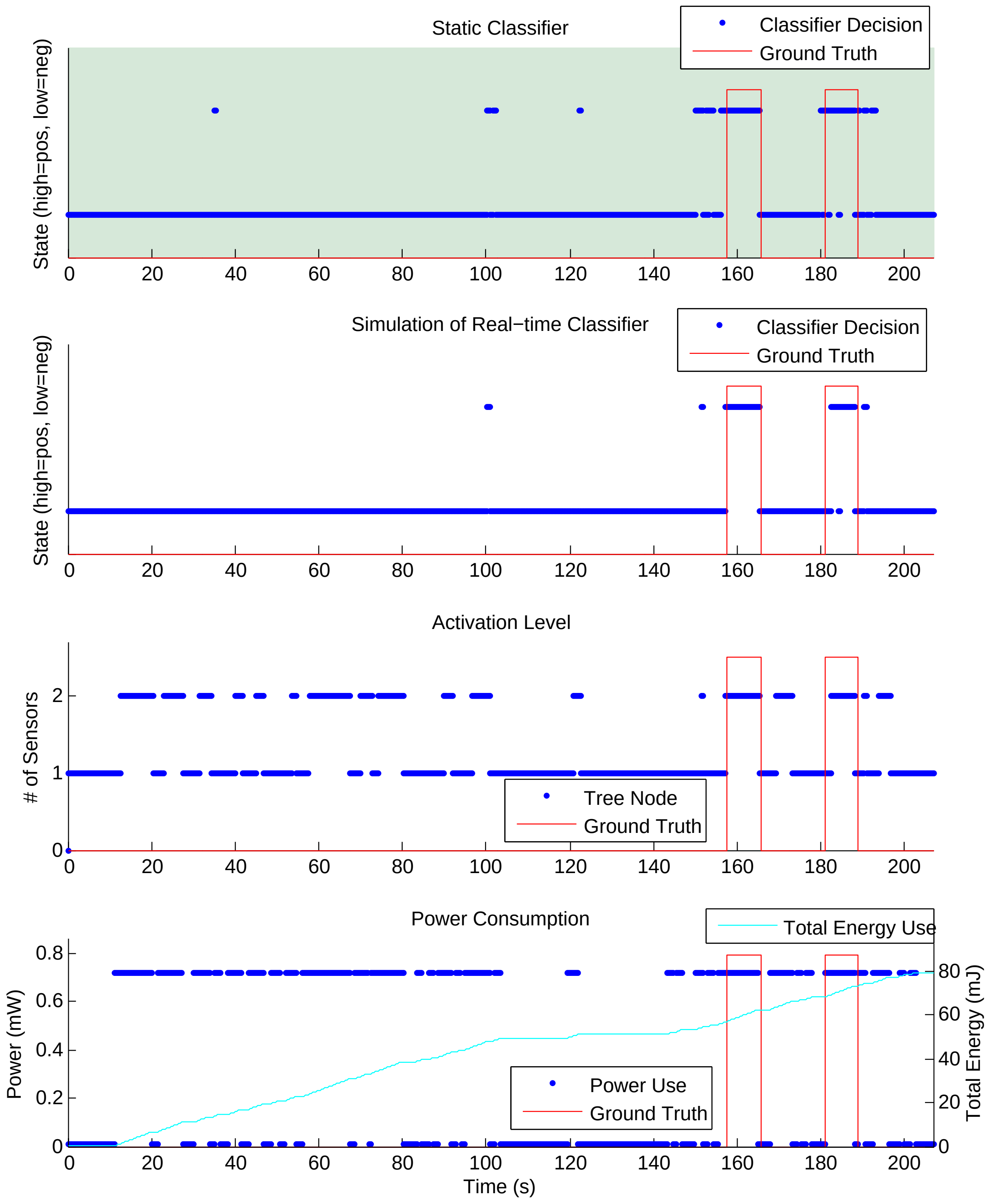


Figure 7-16: Real-time simulation of stair ascent classifier

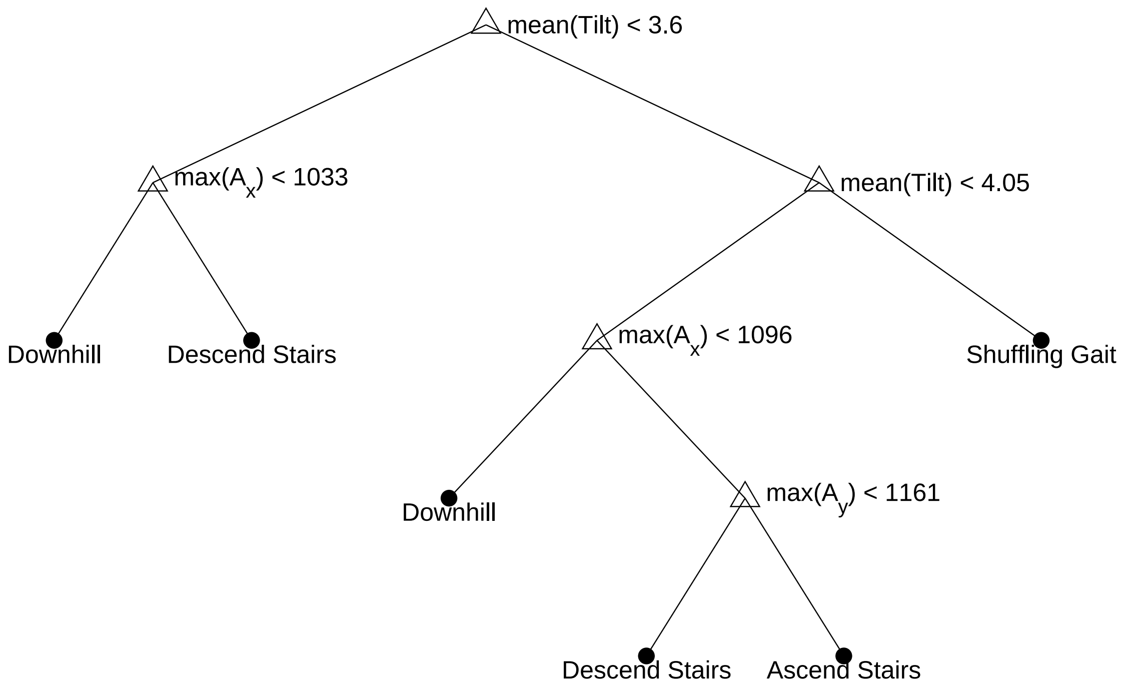


Figure 7-17: Decision tree for stair descent classifier (f=25 Hz)

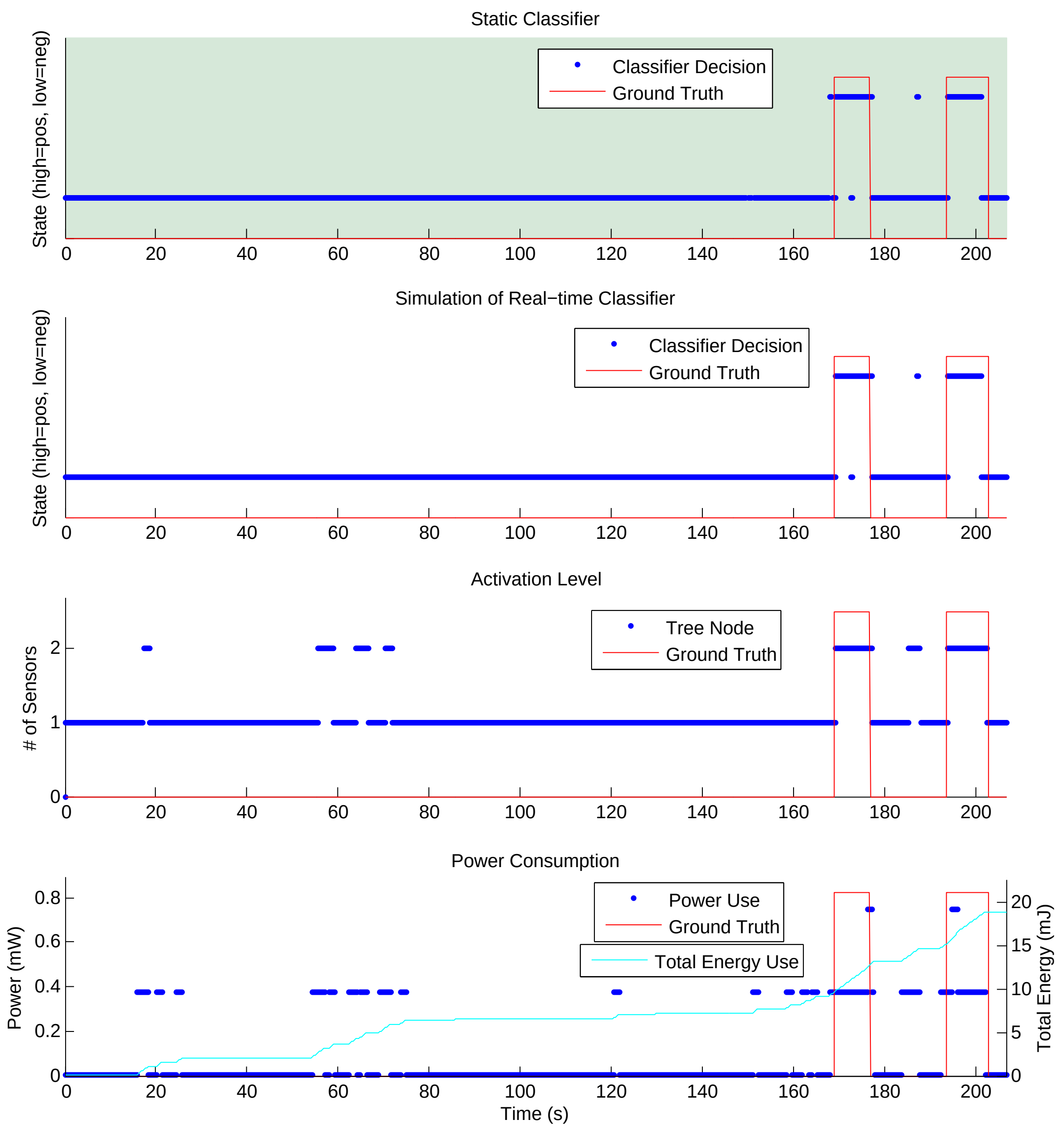


Figure 7-18: Real-time simulation of stair descent classifier

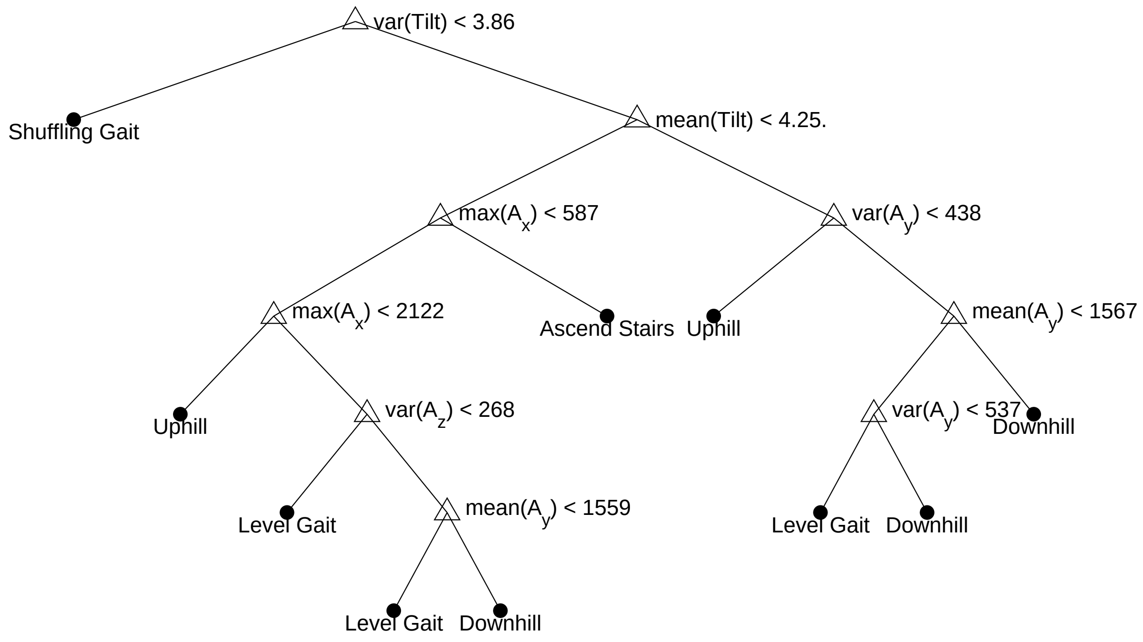


Figure 7-19: Decision tree for level gait classifier (f=200 Hz)

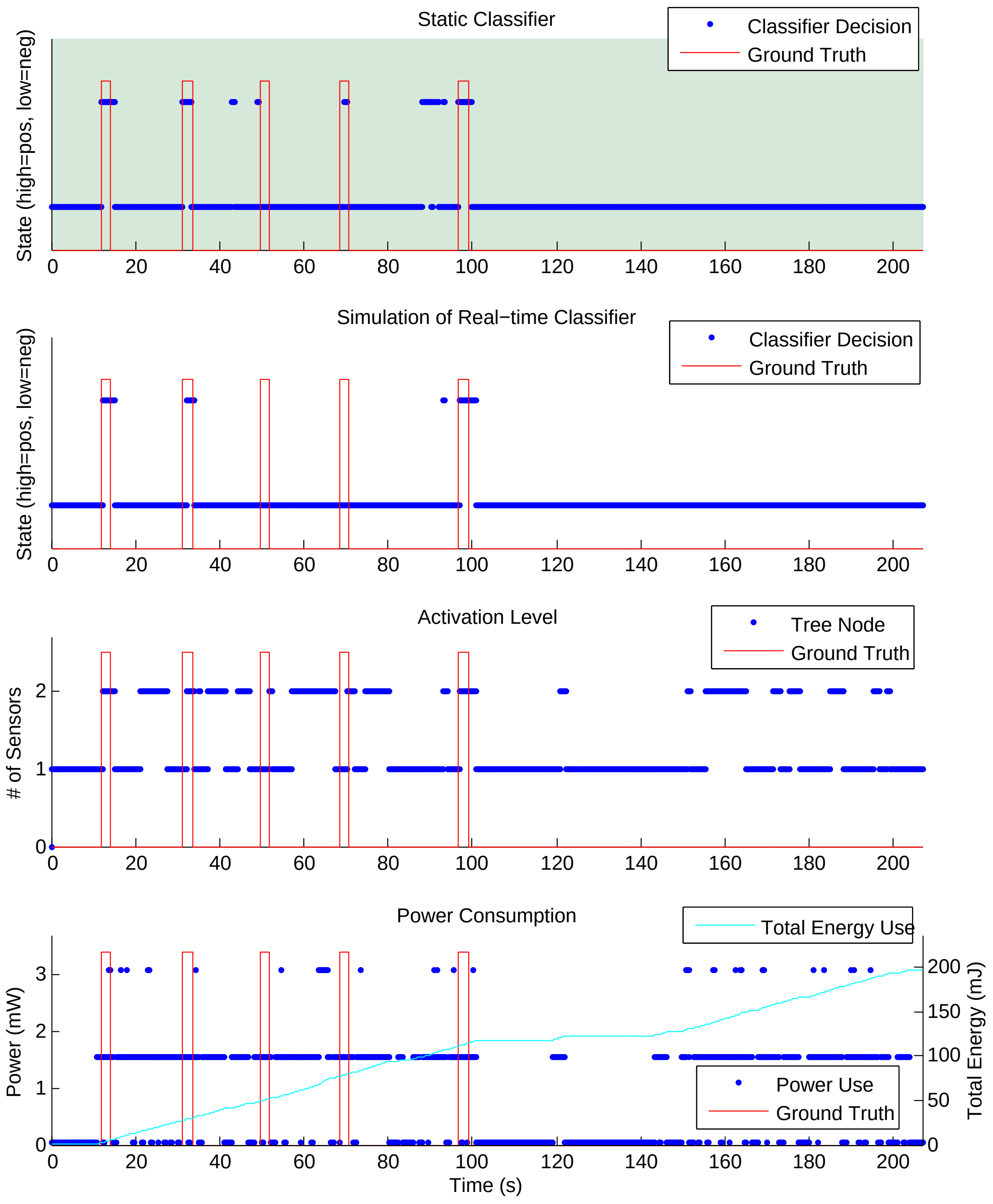


Figure 7-20: Real-time simulation of level gait classifier

Chapter 8

Future Work

In this chapter, we consider some directions for future work on this framework. Specific changes that should be made to the current implementation to improve its functionality and generality are discussed, both overall and with respect to the individual subsystems. Possibilities for expansion of the scope of the framework are also described, focusing on removing the limitations imposed during the design process.

8.1 Improvements to the Implementations

8.1.1 Overall

While the results from the previous section show the benefits of this framework, its overall utility is based on both benefits and ease of use. At present, each of the individual segments is fairly straight-forward for an application designer to use, while the combination thereof can be unwieldy. Data can be collected and downloaded from the hardware fairly easily and software is provided for this purpose. However, the importation into MATLAB is somewhat *ad hoc*. Currently, this is done manually, by converting the time stamps from the recording to array indices, and then tweaking the index manually such that it falls on a state boundary. The final indices are then copied into an array of start and stop values along with state labels. At this point, the construction of the classifier is automated and the creation of different trees for different states or values of W is simply a matter of changing a single line of code. There is, however, no direct way to select the preferred tree based on power and accuracy requirements. The designer must examine the power/accuracy curve produced and manually select the desired tree. Once this solution is found, the embedded code can be automatically generated, leaving the designer the final task of programming the response to the detection of the desired state.

The two transitional steps above can be simplified by providing a user interface to guide the designer. Time stamps from the recording can be easily converted to start and stop indices for each segment[1]. These are then superimposed on the data stream, allowing the designer to tweak their position and then enter a label for the segment. For classifier selection, a simple form allowing the entry of desired power or accuracy and showing the intersection of this value with the curve representing the best classifiers would allow for much simpler selection.

Also of general importance is an examination of the operation of the classifier for an *in situ* or off-body application. The mostly likely domain for a preliminary test would be in the area of wildlife tracking, either outdoor (local fauna around MIT) or indoor (activity in the Media Lab kitchen). This would test the operation of the classification trees with a wider range of sensors (such as the ambient board) and non-periodic data (such as light and heat levels). It would also provide a starting point for examining the use of this framework with wireless networks rather than in individual nodes (see section 8.2.1).

8.1.2 Hardware

While a simple diary-style log served well as a recording medium for the long-term stream (section 7.1), this technique will not always be appropriate. In the case of wearable applications, a subject who is more active, mobile or occupied than in our scenario may well find it

[1]Since this is a supervised training system, acquiring time stamps and labels for the segment are left to the designer.

difficult to keep an accurate log of actions and contexts as required. Therefore, an annotation scheme requiring less attention would be of value. A bank of switches connected to the master board – with each press recorded in the data stream – would allow for easy annotation with little mental effort (beyond remembering the preset mapping between switches and activity types). This hardware could easily be interfaced with the stack through the external interrupt or direct connection lines, or could even comprise its own board. These annotations could be combined with a bootstrapping pattern discovery algorithm (such as [23, 78]) to match marked examples with those not annotated by the user, thereby speeding the labelling process and increasing its accuracy.

In the case of environmental applications, having a human observer keep a log of the events is at best time-consuming, likely disruptive and possibly dangerous (depending on the surroundings). For such application, a passive means of annotation (such as video recording) may be the only feasible approach. However, as noted before, timing skew is likely inevitable over long durations. One solution is to add a real-time clock to the master board. This will allow for orders of magnitude greater timing accuracy by storing the current time in the data stream at regular intervals to keep it in synch with the recording.

In terms of the boards themselves, currently the largest impediment to the use of this framework is that only the IMU board has been modified to allow direct power control over the individual sensors (which was adequate for the examples presented here). Adding this ability to the other sensor boards is slated for the next round of revisions which is currently underway. Also, the ambient board will be modified to add an active lighting system. Most likely in the form of a ring of IR LEDs, this will allow the system to improve the quality of the images taken by the camera by brightening the environment. The magnitude of this response can be based on the sensor data collected by the phototransistors and its power cost will scale with the level of illumination. Other active sensors (*e.g.* proximity monitors) are also under consideration (see section 8.2.3).

8.1.3 Pattern Recognition

The current algorithms for tree construction are agnostic about the desired response to detection. Overall power-efficiency can be increased by combining this response with similar tasks already performed by the system. A data collection task can be combined with the state determination algorithms in several ways, depending on the sensors used in each case. For the sensors sampled within the interesting state, features based on those sensors will have zero cost when used to determine if the system is still in that state. This data collection will include (almost by definition) more sensors with greater net power usage than those used to make the initial detection. Therefore, two trees should be constructed: the one currently built, for use when the system is in an uninteresting state, and a second, for use when the system has been determined to be in an interesting state. An interesting subcase occurs when the data collection involves sensors not included in the construction of the original tree because of high energy usage. One such possibility would be a still camera, which generates large amounts of data per measurement and usually requires complex analysis. These sensors could be included in the second tree for only the marginal cost of the feature calculated from their data (which may well be on par with the total test cost of cheaper sensors).

Since the trees constructed for evaluation were based on a continuously active stream, the use of a single sensor trigger to wake the system from a sleep was not considered. However, in more general applications, this can be of great utility. Such a trigger can be added simply by forcing a single-sided split at the root of the tree. The leaf node selected should be as close to purely negative as possible, since this determination is not even actively considered by the processor. It is possible that this leaf will contain some proportion of the active examples as well, which would increase the power saving even further. The feature used for this split must be a simple binary choice. Therefore, it must come from either a two-state sensor or from a thresholded analog sensor (which is assumed to be low-power[2]). The window size used can be the same as before, with only a single trigger required within that range for the feature to be defined as positive.

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

[2]The LTC1540 micropower ($2\,\mu$W) comparator is suitable for this task

used by 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 current processor should be sufficient in most cases[3]. Similarly, it should be possible to apply knowledge of combinations of sensors which, when used in combination, 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 features, the exact combination must be fixed (*i.e.* the summation $S_1 + \alpha S_2$ for unknown or varying alpha is not acceptable).

The current choice of pruning algorithm is designed to achieve a large decrease in tree size while minimizing the error. However, other goals are possible within the context of detection and merit examination. Reduced error pruning (REP)[32], by its construction, produces a complete ordering of subtrees with respect to error, such that for any tree size (in terms of total nodes) the pruned tree with minimum error is known. This allows for a variety of tradeoffs beyond power versus accuracy, with latency being an obvious choice (as only a subset of the nodes activating a sensor for the first time increase the latency). Also, the REP algorithm can be tweaked such that positive class nodes beyond a certain size are not pruned. While this cannot increase overall accuracy beyond current techniques, it may be possible to increase the performance with respect to certain states, which would prove beneficial for applications with very high misclassification costs.

Also, the current pruning techniques use the error with respect to the pruning set as a metric to determine the right-sized tree. However, it is the error rate on real-time data which is of most interest to the designer. This can be accommodated at two levels. At the lowest level, the error rate of the real-time simulation can be used in place of the risk in the cost-complexity pruning algorithm. This poses some difficulty, since the false positive rate will depend greatly on how conservatively the states are annotated (*i.e.* how many marginal examples on each side are excluded), with the possibility of less accurate trees appearing to be superior in the extreme case. At a higher level, additional pruning of individual subtrees

[3]For example, for the current window size, $O(n)$ algorithms can be applied without taking so long to execute as to cause cycle time difficulties.

with small populations and high marginal test costs can be done. These prunes will be kept if the output state of the decision tree does not change substantially over the annotated (as opposed to transitional) portions of the stream.

8.1.4 Design for an Embedded Implementation

If the trigger variables defined above are used, they must be handled separately before the decision tree can be converted for embedded use. The appropriate sensor outputs need to be connected to one of the external interrupt pins. This input is treated differently depending on the activity level of the system. If the device is at the lowest level - *i.e.* it is waiting for a trigger from this source - then a rising edge trigger is set and the processor is set to sleep. In this state, it is effectively continuously checking the value of the trigger pin. The vast majority of the time, there will be no trigger and the system is assumed to be in the negative state (which requires no response). When a trigger occurs, the system is awoken and moves to the next node in the tree. However, it is still necessary to test if the trigger condition is still met. The method of doing this depends on the form of the trigger. If it is state based, then the direction is reversed to falling edge mode with the interrupt turned off (the associated flag will still be raised). When the flag is seen, the system falls back to the root node and into the sleep state again. If it is edge-based (*e.g.* a number of edges can be expected when the system in a higher activity level), the trigger is left as is but the interrupt is disabled. If the flag is not seen during a cycle, the system falls back to the root node and into the sleep state again[4]. In either case, the root node (and its child leaf) are dropped from the MATLAB tree before it is used to generate the data structure above.

In the case of high tree or state response execution times, it is still possible to construct an embedded implementation with the appropriate timing to allow for power cycling of the sensors. The key is to move the evaluation of the tree from the end of the cycle in which the sensor data is collected to the beginning of the next cycle, overlapping with the sensor wakeup period. This will require interrupt driven toggling between the tree execution and

[4]Some hysteresis may be necessary in this process.

sensor management, but any difficulties are likely minor. Also, there will be an increased latency in tree response, but for high sampling rates this is not a concern.

In our current prototype systems, the only reasonable way to control the power switches was with the microcontroller. However, with application-specific hardware, it should be possible to use an ultra-low power ASIC to control the activation and deactivation of the sensors, such that the microcontroller only needs to wake up to collect the data and execute the classifier. Such an ASIC would need to have a number of different modes (one per activity level) which would be set by the microcontroller in response to the output of the decision tree.

As for the selection of the decision tree itself, it should be noted that there is no reason why this choice need be static. While it is assumed that the application designer will know the desired lifespan of the system to be built and can therefore choose a tree with an appropriate power draw, this is in fact not always the case. In the case of cell phones, the calling level of the owner cannot be known in advance, and in fact varies quite widely. To be able to achieve the same mean time lifespan regardless of user, the system can monitor the average power usage and switch to a less expensive (and, admittedly, less accurate) tree if necessary. Since the tree data structure is quite compact, multiple classifiers can be stored in a single system with little difficulty.

8.2 Expansion of Scope

The limitations described in sections 1.2.3 and 2.3 provide a starting point for the extension of this research, with the end goal of creating a framework for the creation of power-efficient sensor systems which is both general and powerful. We will concentrate on three main avenues. The first are modifications to extend this work to networked sensor systems, thereby allowing for larger and more complex sensing applications. The second are strategies for transforming the pattern recognition into an unsupervised solution, thereby allowing new states to be added over time through use. Finally, the use of active sensors will be touched upon.

	Static	Moving
Corr. Length $= \infty$	Global	
Corr. Length $>$ Spacing	Spatial Gradient	Tracking (1)
Corr. Length $<$ Spacing	Independent	Tracking (2)

Table 8.1: Summary of categories of sensor network problems

8.2.1 Networked Sensor Systems

We divide the application space of networked sensor systems along two axes. The first considers the presence of the phenomena to be sensed – either static or moving. The second considers the correlation length of the phenomena – either infinite (*i.e.* constant across the network), greater than the node spacing or less than the node spacing. Table 8.1 enumerates and names these cases.

In each case, 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. Note that while most sensor networks fuse data to get the best possible result, our goal is simply to get enough information to make a decision on a tree node. Sensor nodes will 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, with the choice based on the details of the case. In general, we believe that considering low-level real-time information in the analysis should allow for better (more accurate and/or energy efficient) solutions.

Measuring global phenomena proves straight-forward. If the nodes are homogeneous, this degenerates into the single node solution with data being collected from as many individual nodes as necessary to make the determination with the desired accuracy. Node usage should also be cycled randomly over time to balance workload, similar to cluster head rotation[71]. For heterogeneous nodes, the problem becomes more interesting. In each state, different nodes will give more accurate results and/or draw less power and the network as a whole should direct its resources towards them. A centralized solution is most efficient here. A second tier decision tree could be constructed, with the features being the state of the individual nodes. The central master would then activate nodes as necessary to most efficiently determine the overall state at a given instant. Deshpande[24] demonstrates a static solution to this problem - *i.e.* power is minimized assuming no knowledge of the individual node response.

When measuring a phenomenon which varies over the network, power savings can be achieved through a reduction in sampling proportional to the correlation[5]. In the simplest case, identical nodes running the same software collaborate on sensing tasks by sharing their measurements with neighbouring nodes. With knowledge of the spatial correlation function, this data can be combined with local data to provide a more accurate estimate. Nodes will only use external info if the marginal cost of reception (beyond the communication already necessary to the network) is the most efficient source of additional information. This is most likely to be the case in networks which communicate continuously (*e.g.* to maintain synchronicity of timers) or for high power sensors (*e.g.* sonar). For heterogeneous nodes, low-level knowledge of all available sensors is necessary for the tree construction, where the possibility that some of those sensors may not be in RF range must be considered. Pradhan[96] demonstrated power savings in this domain by sharing only the bits likely to differ between the node measurements based on a general system model.

In the tracking problem, 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

[5]For a correlation length less than the node spacing, the nodes are independent and the problem degenerates to the single node solution.

by Zhao[130], 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[49] examines 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.

So far, this analysis has assumed that all of the node were collaborating on a single task. However, our techniques could also be extended to sensor networks in which the individual nodes (or groups thereof) were each trying to accomplish different goals. Each task would have a different decision tree to determine the relevant states. Information exchange between nodes with the same goal would most likely be at the state level, while nodes with different goals would most likely communicate low-level sensor readings.

8.2.2 Unsupervised Solutions

Unsupervised training of these sensor systems 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 a good 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 (or even predict). The second benefit would be the potential of the system to add states over time as the user himself changes (*e.g.* the appearance of a shuffling gait). In fact almost any new movement state would be an important physiological marker in and of itself[6].

[6]Assuming minimal false positives in the state discovery algorithms.

While more general and less taxing on the designer, unsupervised training has the drawback of creating unlabelled states, making both real-time and offline analysis more difficult because of the lack of context. Further, to achieve the above benefits, the training must be online, as we hope to quantify ephemeral and emerging states. To allow the system itself to remain relatively low power, heuristics for the occurrence of new or interesting events will need to be created. Given 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 state. Given information from (for example) the user's calendar (or other contextual source), transitions between (and sometimes the content of) the entries can be used as cues that the sensor system should be in a high complexity state[7]. In both cases, full data streams can be collected for analysis and comparison to known states. If they are significantly different, a new state is added. It may be possible to label this state based on the above calendar information, or the users themselves could be cued to answer questions about the state when the system has access to a GUI (most likely during a data download, as done in [54]).

As a further step to possibly capture very short states (which may be missed as the system powers up consecutive levels of sensing), 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 it being added to the states in the decision tree. This method will take a very long time to (effectively 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. Given knowledge of possible circumstances which would drastically alter the state detection (*e.g.*, for the wearable gait lab, a sharp increase in outdoor humidity and/or temperature might correlate to reduced activity levels), low-power static sensors with (high) preset threshold can be added to detect these cases for nominal reduction in battery life[8].

[7]As defined by the amount of sensing, *i.e.* the state's leaf node depth

[8]See [72] for one such system.

8.2.3 Active sensors

The difficulty of incorporating active sensors is that it adds an order of dimensionality to the pattern recognition algorithms, as the variation of output power affects the received signal. Currently, data is collected for training at the maximum accuracy for each sensor and assumes a fixed energy usage per sample. While passive sensors represent a single point on the power/accuracy plane, the various output states of active sensors form a (most likely non-linear) functional relation. We consider two possible cases. Range-based sensors (*e.g.* sonar) will tend to have a step-shaped power/detection accuracy curve. Therefore, the sensors can be considered to have only a single useful operating point - the one just beyond the threshold. While this point will vary with the range of the target, it should be possible to produce a reasonable estimate based on the details of the state in which it is being used. This case is then equivalent to that of a passive sensor. Environmental sensors (*e.g.* IR illuminated camera) will have a more complicated power/accuracy curve. In the best case scenarios, there will be a number of local minima, each of which can then be considered as a separate sensor for the purposes for the pattern recognition algorithms. For flatter curves, a few salient points can be chosen by the designer based either on the properties of the other sensors available to measure the same phenomena or on the expected system states. During training, the tree will choose the most appropriate operation point (from those selected above) based on the accuracy, power usage and the other sensors' data. Note that since active sensors tend to use far more power than the embedded sensors currently considered, they are likely to be used only if they are the only sensor which can distinguish between a collection of training examples.

Chapter 9

Conclusions

9.1 Overview of Work

This dissertation has presented a three-component framework for power-efficient detection in wearable sensors. The first is modular hardware platform for ease of application prototyping. As opposed to similar architectures, this system treats the sensor boards as discrete design objects that have data collection as their primary goal. Each of the boards within this platform encapsulates design knowledge of the best practises within an area of sensing (*e.g.* inertial measurement). The architecture is currently being adapted to better reflect low-power goals. This entails the addition of power switches to individually control the sensors, replacement of amplifiers with more efficient models, and the updating of various sensors to take advantage of improvements in the underlying technology.

The second and key component is a semi-autonomous classifier construction algorithm. Supervised training was chosen to allow for designer selection of the particular states of interest. Using a sensor node constructed with the modular hardware platform, a scripted data stream is collected and annotated to provide suitable data for training. A set of simple and efficient features - minimum, maximum, mean and variance (and their robust versions) - are used for generality, though others can be added. The features are calculated over a fixed window (based on the period of the data), thereby converting the data stream into a set of examples to be used by a classical machine learning 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 (or subsets thereof), with some requiring far fewer decisions (and therefore far less energy) than others. The standard top down induction algorithm for decision trees is 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. For a given activity level, sensors are powered up in the inverse order of their wakeup time, such that they all come online at the same time for sampling by the processor. Recursive implementation of the selected features allows for efficient calculation thereof from the collected data. The tree classifier is then executed as a series of simple comparisons. The state output and request for activation or deactivation of sensors based on the result of the tree execution are smoothed to avoid expensive spurious actions.

This framework was analysed in a number of ways. Examples of successful sensor nodes built from the modular hardware platform are presented. The features chosen for use in the classifier were shown to be consistent across a range of window sizes within their labelled state. Tests with three popular data sets demonstrated the monotonicity of average test cost with the weighting parameter. To examine one specific case in depth, a wearable gait monitoring system was built using inertial measurement. Data was collected for six different motions and classifiers were constructed to separate each one from the other five. These classifiers performed better on the power/accuracy plane over their range than a support vector machine (SVM) trained for the same task[1]. Tree classifiers built without the benefit of redundant sensing (in this case, tilt switches to shadow the accelerometer) performed

[1]Note that Gaussian SVMs are computationally unsuitable for embedded applications.

less well than those which had both modalities available, but still outperformed the SVM. Simulation of real-time operation demonstrated that the tiered system drew substantially less power than a single-trigger (binary) wakeup system, with only a negligible increase in latency.

9.1.1 Summary of Application Designer Tasks

Because of the user-driven nature of this framework, a number of tasks are explicitly left to the application designer as part of this process. While the application designer will, of course, need to specify the states to be detected, other requirements are less obvious. These are itemized below:

- Selection of the sensors necessary to measure the states of interest
- Prototyping of a node containing those sensors using The Stack
- Determination of states similar to those to be detected
- Collection and annotation of a data stream containing the desired (and similar) states
- Selection of the desired power/accuracy operating point
- Designation and coding of the desired response to the state of interest

Of the above tasks, only the first and last are explicitly technical. While those with a general familiarity with electronics may have an intuition as to which sensors are appropriate for their task, some guidance will likely be necessary. The programming of the desired response is a more difficult matter, as facility with coding for desktop computers does not translate to the ability to write code for embedded applications, which often requires low-level design knowledge. This suggests that a selection of simple responses should be added to the code base of The Stack.

9.1.2 Assumptions

Assumptions of a human-centric application (specifically the wearable gait monitor system test) were used at various points in the design process. The sensor choices, as well as their sampling rates and accuracies, were thereby limited. Similarly, the analysis functions used to create the decision tree were constrained. An explicit listing of all the assumptions is:

- Sampling rate range of 25 Hz to 200 Hz
- Data period of ~ 1 sec
- Minimum action length of ~ 10 sec
- Utility of the windowed mean, variance, maximum and minimum
- Use of non-restrictive misclassification costs

While it is unlikely that generalizing these assumptions will create major difficulties, the risk are worth discussing.

The restriction on the sampling rate simplifies the construction of the embedded software by allowing the classifier to simply run at the maximum sample rate of the system. A higher maximum (such as for audio sampling) could result in an update rate faster than any possible state change and therefore in wasted power. The combination of the data period and the minimum action length allows for classifiers with depths around five without the risk of missed states (taking into account that the classifier will rarely jump from the lowest level to the highest). Ephemeral events - those with periods very close to their length - cannot be detected by this system. Similarly, the use of windowed features assumes either periodicity or finite correlation length. Otherwise, the assumption of consistency across the labelled classes will not hold and the classifier will increase in complexity. Finally, there are two key assumptions about the misclassification costs. First, that they are not highly skewed towards either positive or negative examples, which can make the training of the decision tree very difficult, as a small number of examples suddenly hold large weight. Second, that misclassification costs in the units of energy are an order of magnitude greater than the cost of determining the state. Since this system has no restrictions on total cost, it is possible that a classifier could be constructed for which the cost of determining the state is greater than the penalty paid for being incorrect.

9.2 Detailed Contributions

The core contribution of this dissertation is an automated framework allowing application designers to easily construct power-efficient state detection systems. The following are the main parts of that framework:

- A modular hardware platform centred on wearable sensing application was built and used to construct a number of sample applications. The platform incorporates a number of low-power design techniques to aid in the construction of efficient wearable and embedded sensors.
- A multiplicative test cost weighting was added to the standard decision tree training algorithm. The form of the weighting is justified and allows for zero cost tests.
- The weighting parameter was shown to alter the growth of the decision tree such that higher values of the parameter will result in less costly trees.
- In a complex sample application, these classifiers were shown to perform better on the power/accuracy plane over their range than a support vector machine (SVM) trained for the same task.
- The addition of redundant sensors allowed for better performance in the bottom left corner of the power/accuracy plane which would otherwise have not been possible.
- Simulation of real-time operation demonstrated that the tiered system drew half as much power as a single-trigger (binary) wakeup system with only a negligible increase in latency.

Appendix A

Other Applications Using The Modular Sensor Platform

Because of space and flow constraints, not all of the applications constructed or prototyped with The Stack could be listed in chapter 4. They are presented here for completeness.

A.1 Constructed Applications

A.1.1 Roballet

Roballet was an initiative of the Future of Learning group at the MIT Media Lab to introduce children to dance and choreography. The goal was to allow them to experience music as a space to be explored, rather than a set of rules to conform to. Larissa Welti-Santos led a group which used the modular hardware, in a configuration similar to that of the wearable gait laboratory(section 4.5.1), to measure the children's motion in real-time[124].

The Stack was mounted on the dancer's upper arm, as seen in figure A-1a, with the IMU board capturing whole body motion. Rather than using the tactile board with an insole as before, the children each constructed a glove using the bend and pressure sensors (figure A-1b). Two musical pieces were designed, with the sensors controlling lighting and animation.

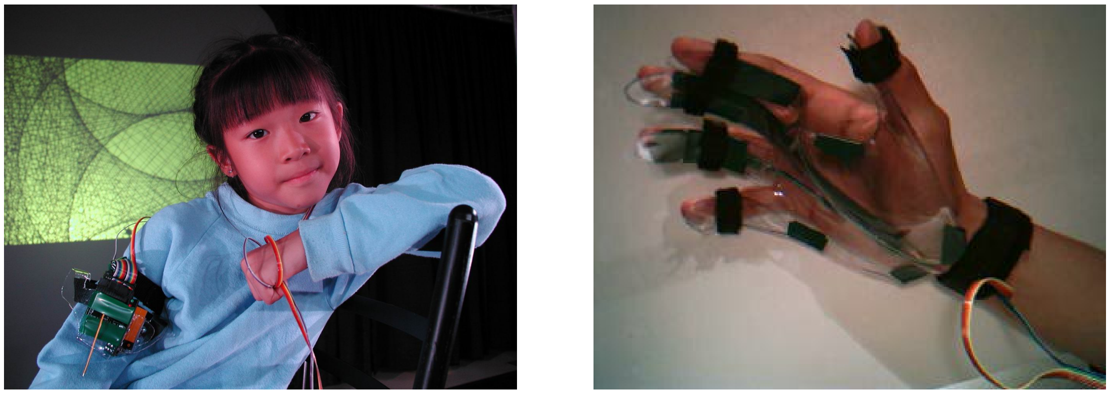

(a) Child wearing main sensor (b) Glove

Figure A-1: Roballet sensors

In the first, information from the flex sensors was used to change the lighting and background animation during the dance. In the second, the pressure sensors were used to cue the animation while the motion sensors were used to control its speed. Unfortunately, the pieces were not performed as such because of complaints by the choreographer about the "visibility" of the sensors.

This project benefitted greatly from the ease of prototyping with the system. The IMU board was used without modification, and provided useful inputs to the dance applications. The tactile board proved even more flexible, allowing children with little knowledge of electronics to quickly and easily build a sensor glove. This glove could interface with the rest of the system with no redesign of either hardware or software. Overall, use of The Stack allowed the low-level hardware details to be abstracted away, such that only the high-level details of the motion and pressure data being collected was of concern to the choreographer and the children.

A.1.2 Novel Musical Controller

Another application implemented using this platform is a trainable adaptive musical controller [75]. In traditional musical instruments, each input gesture is connected to a specific output sound through the laws of physics (which are cleverly manipulated to create the desired effect). The advent of digital sound synthesizers and electronic music controllers has

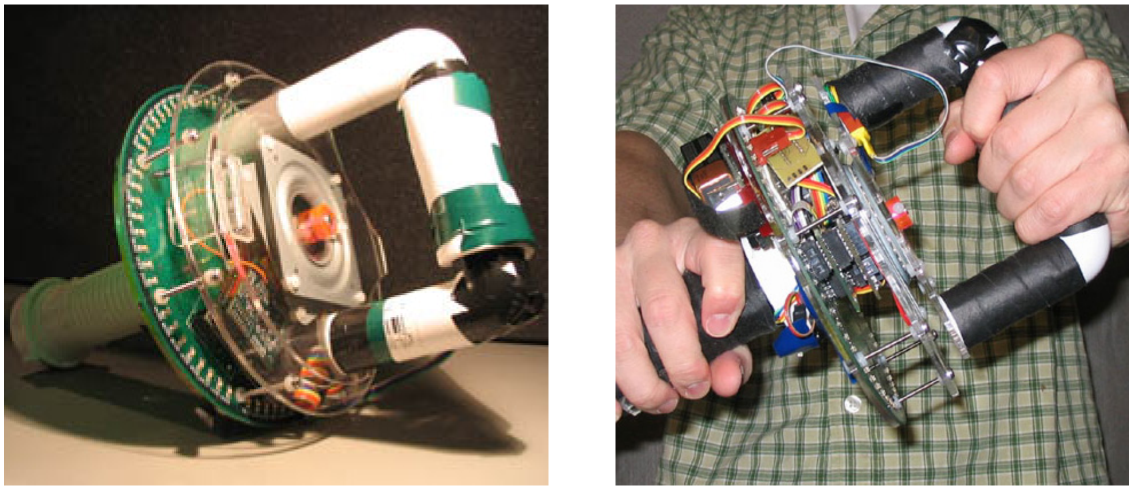


Figure A-2: FlexiGesture controller

decoupled action and reaction, making a large range of input-to-output mappings possible, as well as an infinite set of possible timbres. However, this revolutionary change brings with it the problem of design - intuitive, natural mappings from gesture to sound now must be created in order to create a playable electronic music instrument.

FlexiGesture, created by David Merrill, is a device that allows flexible assignment of input gesture to output sound, thus acting as a laboratory to help further understanding about the connection from gesture to sound. It is controlled in a two-handed fashion, with a main handle and twistable top as shown in figure A-2. Manipulations of the device relative to itself (bending, twisting, *etc.*) are measured using the tactile board. Movement of the device in the world frame is measured using the IMU board. The data is collected and transmitted using the master board. Two new boards were also created. An output pane was built with circuitry to control a pager motor and LEDs for user feedback as well as transmit circuitry for electric field position sensing[88]. An electric field receiver board was built to detect these signals, and formed part of the basestation (nominally comprised of a just master board).

We note two main benefits of the system here. The first is the obvious exploitation of the extensibility, as described above. The second is the benefits derived from encapsulation, which allowed the designer to reuse the circuitry of the IMU board, the master board (twice) and the tactile board in a new arrangement. This resulted in a significant reduction in design time for the electronics, allowing the designer to concentrate on the interaction aspects of the project rather than the data collection.

A.2 Prototyped Systems

Not all applications use the boards from this platform for their final implementation. Often our boards are used simply for prototyping and design, and the circuitry is then redesigned depending on the particular physical constraints of the systems. Two such systems are touched on below.

A forearm controller and tactile display[106], designed by David Sachs, allows the wearer to control a braille output scheme by rotating or raising or lowering their arm. It used components of our platform in its original incarnation. The IMU board was used in the motion tracking system, with its data collected by a different processor than that on the master board. A new board was also created to test the benefits of magnetic sensors for tracking. Eventually, these components were integrated with a previously constructed circuit board which controlled the output portion of the application.

Ryan Aylward has constructed a compact coordinated gesture system[3] for concurrently collecting motion data from a group of individuals at various points on their bodies. This data is then analysed *en masse* for similarities in movements and gestures. The initial application of this system was the instrumentation of a half-dozen dancers at four points each (ankles and wrists), with the output data used to create music on-the-fly. This is a direct extension of the work in the Expressive Footwear project[89] (identical in goal, but with a single dancer) and the wearable gait laboratory (described previously). This system was prototyped using the IMU and master boards, but the hardware was later reconfigured into a package with nearly identical functionality, but planar and one third the height[68].

In both cases, the modularity of the architecture was exploited to quickly prototype new systems. In the case of the forearm controller, we note that once the software to read data from a single board was incorporated into an existing system, it was trivial to have it read data from a second, newly created board. In the case of the gesture system, each of the individual subsystems were tested and revised (both individually and as a whole) as part of the gait laboratory project before the combined system was designed. This new design was vetted more quickly and had a higher assurance of success because of this prior work.

Appendix B

Details of Data Collection

B.1 Sensor Measurement Axes

The coordinate axes for the sensor module are shown in figure B-1.

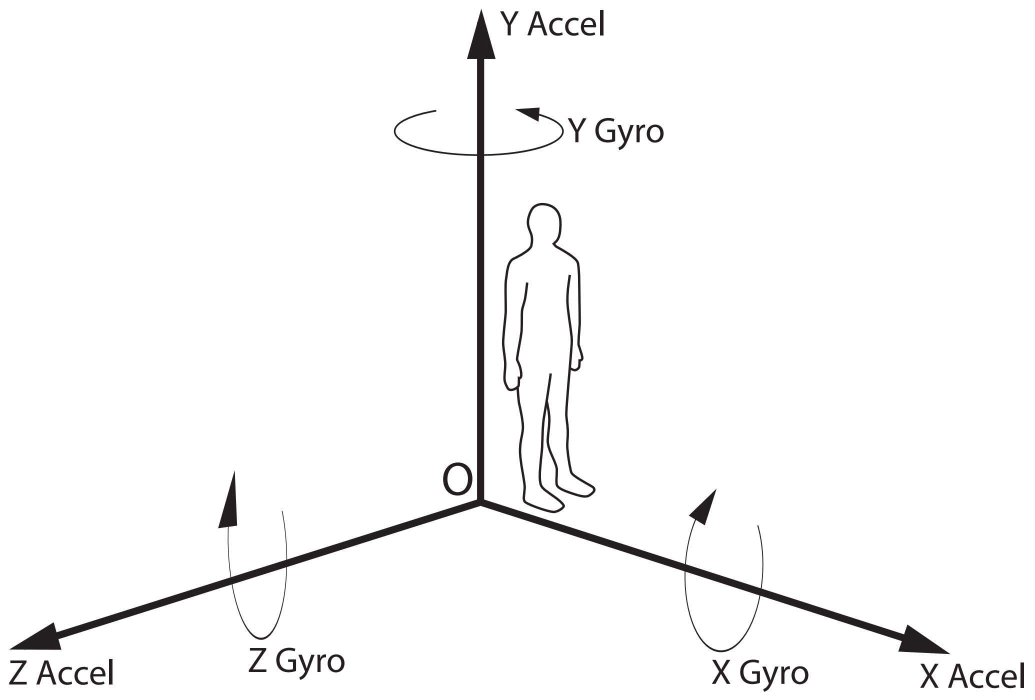


Figure B-1: Coordinate axes for the sensor module

B.2 Stream Used for Classifier Training

The training data stream was collected in the lower lobby of the MIT Media Laboratory. The following motions were collected:

Level Gait — Walking back and forth across the lower lobby four times. Roughly 4 minutes.

Shuffling — Mimicries Parkinsonian shuffling (as per [40]) back and forth across the lower lobby. Roughly 3 minutes.

Ascend Stairs — Climbing the second segment (13 steps) of the stairwell leading from the lower lobby to the atrium nine times. Roughly 3 minutes.

Descend Stairs — As above, except descending.

Uphill — Climbing the $\sim 7.5°$ slope from the bottom of Bartos theatre to the top eight times. Roughly 3 minutes.

Downhill — As above, except descending.

Figure B-2 shows the specific locations where the data was collected.

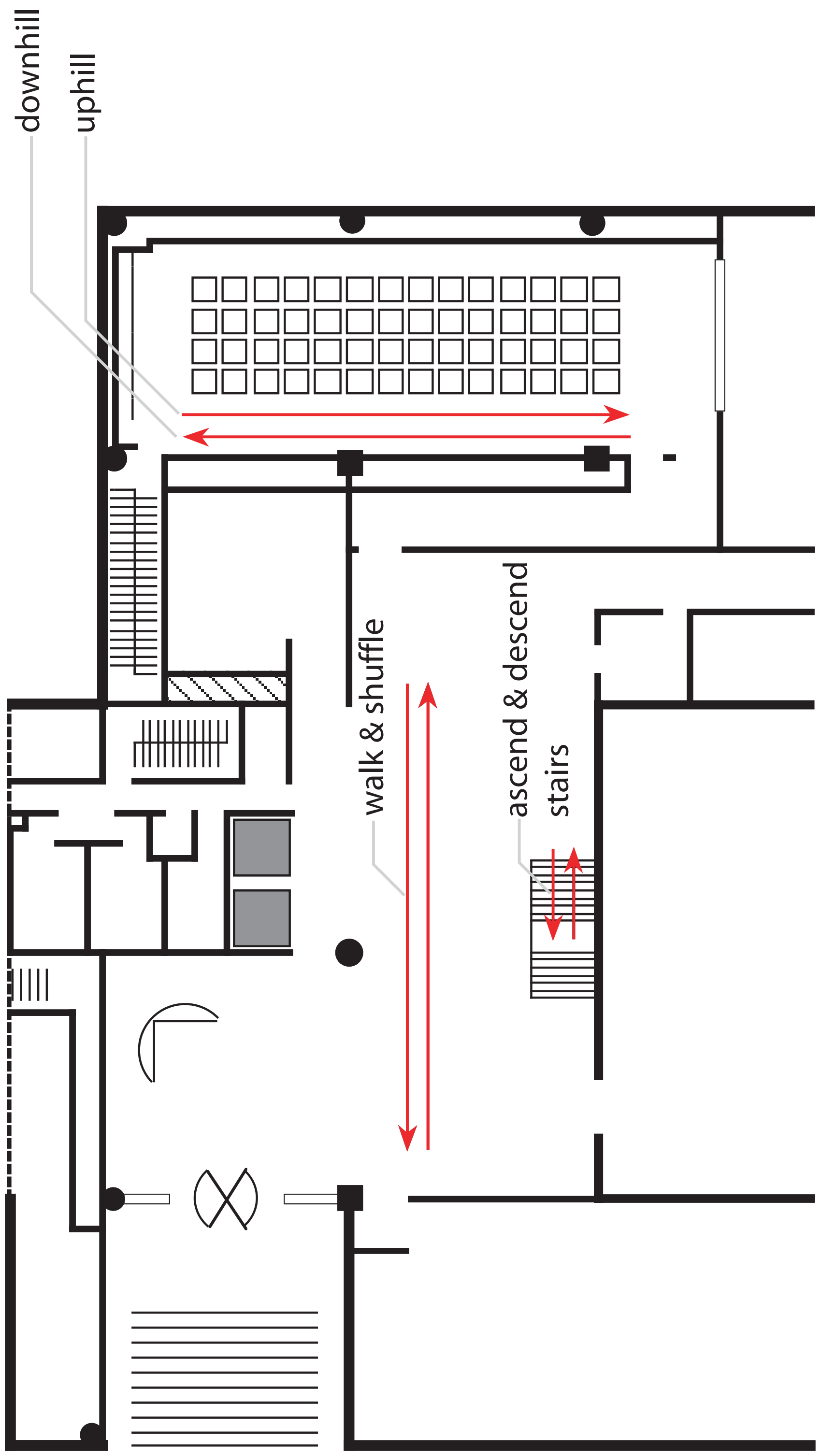


Figure B-2: Locations for training set data collection

B.3 Stream Used for Real-time Simulation

Starting from the entrance of Bartos theatre, the subject performed two loops (level, downhill, level, uphill) before leaving the theatre via a shallow ramp ($\sim 4°$). The subject then walked a short distance into the lower lobby, shuffled for roughly 20 feet, walked for five steps, and then shuffled for another 20 feet. From there, the subject transitioned to the staircase, which was ascended and descended twice. Figure B-3 shows the full path taken. The dotted lines denote unlabelled transitional segments.

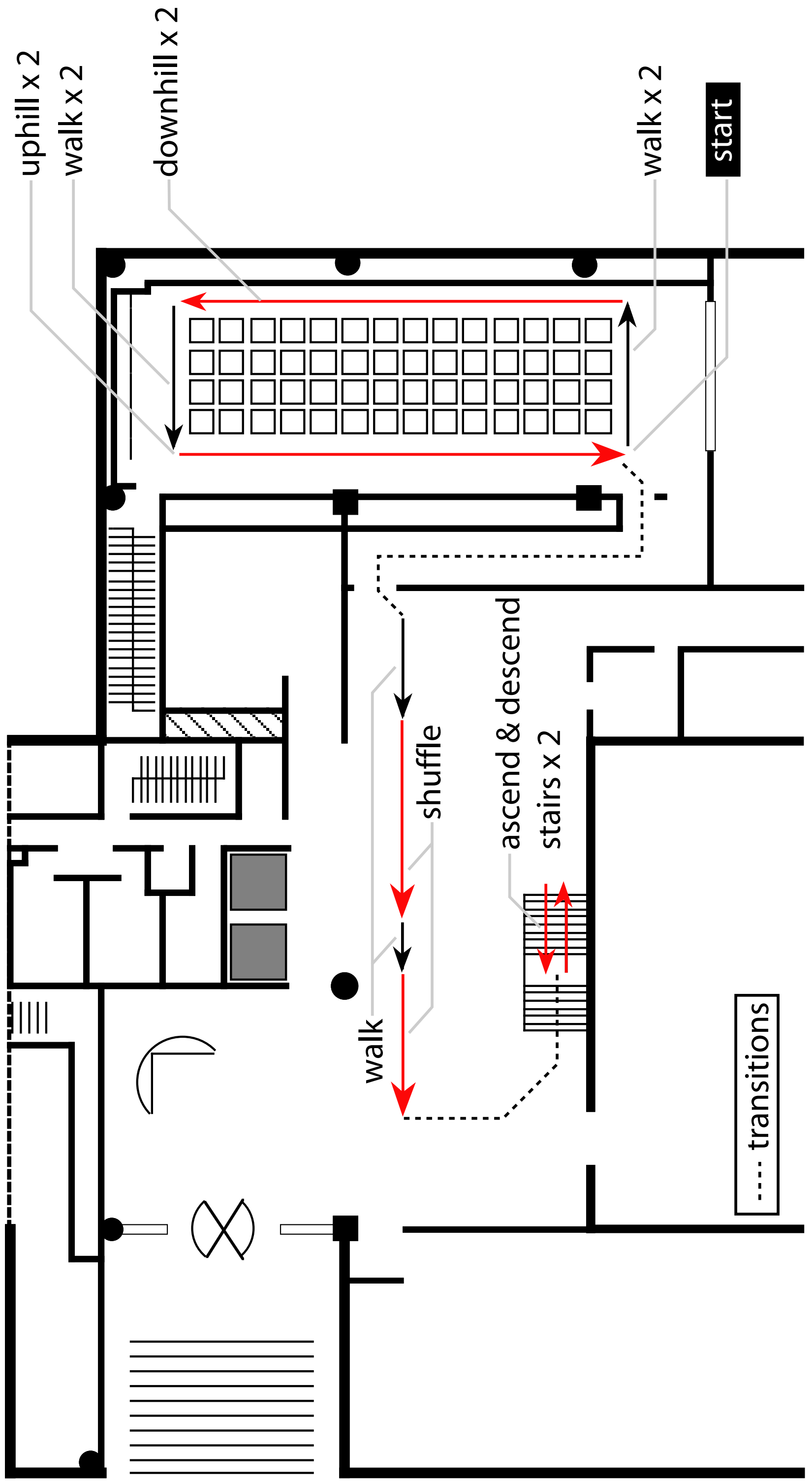


Figure B-3: Path taken for test set data collection

Appendix C

Overview of Changes to MATLAB Decision Tree Implementation

Version 7.1 (2006b) of MATLAB (the current version at the time of this writing) contains a basic implementation of the CART algorithms, including cost-complexity pruning. This implementation was significantly altered to add three new features. The first was test cost handling, including changes to the ordering of possible splits, application of the weighting parameter and discounting of features after first use. The second was the addition of the two new splitting criteria used in the testing. The third was the ability to designate a single positive state in non-binary (> 2) class systems, such that the rest of the classes are considered as a single negative state for the purposes of misclassification cost.

The following portions of the implementation were changed to allow for our testing and tree construction:

- Additions to the decision tree data structure
    - Node level tracking of current:
        - Feature test costs
        - Active sensors
        - Depth and latency
    - Index of positive class

- Changes to `treefit.m`
    - Input of values necessary for test cost handling:
        - Base test costs
        - Grouping of features for discounting
        - Weight parameter W
    - New splitting criteria:
        - AUC-based
        - One-sided Gini
    - Generalized application of test cost to choice of split
- Changes to `treetest.m`
    - Cross-validation and holdout set estimation of:
        - Test cost
        - Latency
        - Depth
- Changes to `treedisp.m`
    - Ability to display test costs when visualizing tree

Since the MATLAB source code is under copyright, the altered versions of the files cannot be reproduced here. Without them, the high-level scripts used to build the population of classifiers and simulate the real-time operation would most likely serve to confuse the reader. For those interested in obtaining copies of the implementation and data, instructions are available at:
http://www.media.mit.edu/resenv/groggy/
or by contacting the author at:
`ayb` AT `alum` DOT `mit` DOT `edu`

Bibliography

[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. (Cited on pages 21 and 55.)

[2] K. Aminian, P. Robert, E. E. Buchser, B. Rutschmann, *et al.* Physical activity monitoring based on accelerometry: validation and comparison with video observation. *Med Biol Eng Comput*, 37(3):304–308, May 1999. (Cited on page 78.)

[3] R. Aylward. *Sensemble: A Wearable Inertial Sensor System for Collective Gesture Tracking and Interactive Dance.* SM thesis, Program in Media Arts and Sciences, Massachusetts Institute of Technology, September 2006. (Cited on page 180.)

[4] L. Bao and S. S. Intille. Activity recognition from user-annotated acceleration data. In *Proceedings of PERVASIVE '04*, LNCS 3001, pages 1–17. Springer-Verlag, 2004. (Cited on pages 51 and 80.)

[5] K. C. Barr and K. Asanovic. Energy aware lossless data compression. In *Proceedings of MobiSys '03*, pages 231–244. April 2003. (Cited on page 44.)

[6] A. Y. Benbasat and J. A. Paradiso. An inertial measurement framework for gesture recognition and applications. In Ipke Wachsmuth and Timo Sowa, eds., *Gesture and Sign Language in Human-Computer Interaction*, pages 9–20. 2002. (Cited on pages 57 and 79.)

[7] A. Y. Benbasat and J. A. Paradiso. Design of a real-time adaptive power optimal sensor system. In *Proceedings of IEEE Sensors '04*. October 2004. (Cited on page 34.)

[8] A. Y. Benbasat and J. A. Paradiso. A compact modular wireless sensor platform. In *Proceedings of IPSN '05*, pages 410–415. April 2005. (Cited on pages 23 and 57.)

[9] Berkeley Wireless Embedded Systems Project. http://webs.cs.berkeley.edu/. (Cited on page 32.)

[10] M. Bhardwaj, R. Min, and A. Chandrakasan. Quantifying and enhancing power awareness of VLSI systems. *IEEE Transactions on VLSI*, pages 757–772, December 2001. (Cited on page 54.)

[11] A. P. Bradley. Use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997. (Cited on page 88.)

[12] L. S. Brakmo, D. A. Wallach, and M. A. Viredaz. μSleep: a technique for reducing energy consumption in handheld devices. In *Proceedings of MobiSys '04*, pages 12–22. June 2004. (Cited on page 56.)

[13] L. Breiman. Bagging predictors. *Machine Learning*, V24(2):123–140, 1996. (Cited on page 86.)

[14] 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. (Cited on page 83.)

[15] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984. (Cited on pages 86, 89, 95, 96, 104 and 105.)

[16] E. O. Brigham. *The fast Fourier transform and its applications*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1988. (Cited on page 79.)

[17] S. Burke and R. Poor. Using wireless sensor networks to keep watch over the nation's bridges. In *Sensors Expo and Conference*. 2004. (Cited on page 32.)

[18] C-C. Chang and C-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/c̃jlin/libsvm. (Cited on page 131.)

[19] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002. (Cited on page 90.)

[20] S. Cheekiralla. *Wireless Infrastructure Monitoring*. SM thesis, Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, February 2004. (Cited on page 32.)

[21] C. Chien, M. B. Srivastava, P. Lettieri, V. Aggarval, *et al.* Adaptive radio for multimedia wireless links. *IEEE Journal on Selected Areas in Communications*, 17(5):793–813, May 1999. (Cited on page 52.)

[22] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989. (Cited on page 90.)

[23] B. Clarkson. *Life Patterns: structure from wearable sensors*. PhD thesis, Program in Media Arts and Sciences, Massachusetts Institute of Technology, September 2002. (Cited on pages 29 and 161.)

[24] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, *et al.* Model-driven data acquisition in sensor networks. In *Proceedings of the VLDB '04*, pages 588–599. 2004. (Cited on pages 51 and 167.)

[25] T. G. Dietterich. Machine learning for sequential data: A review. In *Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, pages 15–30. Springer-Verlag, London, UK, 2002. (Cited on page 80.)

[26] J. P. DiMarco. Implantable cardioverter-defibrillators. *New England Journal of Medicine*, 349(19):1836–1847, 2003. (Cited on page 28.)

[27] R. Duda, P. Hart, and D. Stork. *Pattern Classification.* Wiley, 2nd edition, 2001. (Cited on pages 82 and 94.)

[28] P. Dutta. Presentation at IPSN '05. (Cited on page 31.)

[29] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, *et al.* Design of a wireless sensor network platform for detecting rare, random and ephemeral events. In *Proceedings of IPSN '05.* April 2005. (Cited on pages 31, 49 and 50.)

[30] F. Eady. E-field evaluation module. *Circuit Cellar*, pages 60–66, June 2003. (Cited on page 65.)

[31] A. Ephremides. Energy concerns in wireless networks. *IEEE Wireless Communications*, pages 48–59, August 2002. (Cited on page 53.)

[32] F. Esposito, D. Malerba, G. Semeraro, and J. Kay. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, May 1997. (Cited on pages 87, 94, 95 and 163.)

[33] D. Estrin, D. Culler, D. Pister, and G. Sukhatme. Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, pages 59–69, January-March 2002. (Cited on page 47.)

[34] ETA SA. ETA Trendline F05.111 Specifications. https://secure.eta.ch/CSP/DesktopModules/ViewDoc.aspx?DocId=129&DocType=DT. (Cited on page 24.)

[35] M. Feldmeier and J. A. Paradiso. An interactive music environment for large groups with giveaway wireless motion sensors. *Computer Music Journal*, 31(1), 2007. (Cited on page 33.)

[36] C. Ferri, P. Flach, and J. Hernández-Orallo. Learning decision trees using the area under the ROC curve. In *Proceedings of the 19th International Conference on Machine Learning*, pages 139–146. 2002. (Cited on page 88.)

[37] C. Ferri, P. Flach, and J. Hernández-Orallo. Improving the AUC of probabilistic estimation trees. In *ECML 2003: 14th European Conference on Machine Learning*, volume 2837 of *Lecture Notes in Computer Science*, pages 121–132. 2003. (Cited on page 94.)

[38] F. R. Finley and K. A. Cody. Locomotive characteristics of urban pedestrians. *Arch Phys Med Rehabil*, 51(7):423–6, 1970. (Cited on page 97.)

[39] Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995. (Cited on page 87.)

[40] GaitBrowser: Pathological Gait Viewer. http://guardian.curtin.edu:16080/cga/GaitBrowser/. (Cited on pages 123 and 184.)

[41] V. Garg and J. Wilkes. *Wireless and Personal Communications Systems*. Prentice Hall, 1996. (Cited on page 55.)

[42] A. Gelb, ed. *Applied Optimal Estimation*. MIT Press, 1974. (Cited on page 48.)

[43] J. Gemmel, L. Williams, K. Wood, R. Lueder, *et al.* Passive capture and ensuing issues for a personal lifetime store. In *Proceedings on CARPE '04*, pages 48–55. October 2004. (Cited on page 30.)

[44] S. Glaser. Some real-world applications of wireless sensor nodes. In *Proceedings of SPIE Symposium on Smart Structures and Materials*. March 2004. (Cited on page 32.)

[45] T. Gorton. *Tangible toolkits for reflective systems modeling*. SM thesis, Program in Media Arts and Sciences, Massachusetts Institute of Technology, June 2003. (Cited on page 69.)

[46] A. Grover. Modern system power management. *Queue*, pages 66–72, October 2003. (Cited on page 56.)

[47] D. J. Hand and R. J. Till. A simple generalisation of the area under the roc curve for multiple class classification problems. *Machine Learning*, 45(2):171–186, November 2001. (Cited on page 88.)

[48] T. Hastie, R. Tibshirani, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer-Verlag, 2001. (Cited on page 105.)

[49] T. He, B. Krogh, S. Krishnamurthy, J. A. Stankovic, *et al.* Energy-efficient surveillance system using wireless sensor networks. In *Proceedings of Mobisys '04*, pages 270–283. ACM Press New York, NY, USA, 2004. (Cited on pages 49, 50 and 168.)

[50] J. Hightower, A. LaMarca, and I. E. Smith. Practical lessons from Place Lab. *IEEE Pervasive Computing*, 5(3):32–39, July-Sept. 2006. (Cited on page 33.)

[51] J. Huang and C.X. Ling. Using AUC and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):299–310, Mar 2005. (Cited on page 88.)

[52] S. S. Intille. Designing a home of the future. *IEEE Pervasive Computing*, 1(2):76–82, April 2002. (Cited on page 19.)

[53] 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. ACM Press, New York, NY, USA, 2004. (Cited on page 75.)

[54] S. S. Intille, E. Manuel Tapia, J. Rondoni, J. Beaudin, *et al.* Tools for studying behavior and technology in natural settings. In *Proceedings of UbiComp '03*, pages 157–174. Springer-Verlag, October 2003. (Cited on pages 33 and 169.)

[55] iPod Battery FAQ. iPod Battery and Power Specifications. http://ipodbatteryfaq.com/ipodbatteryandpower.html. (Cited on page 24.)

[56] A. Jain and E. Chang. Adaptive sampling for sensor networks. In *Proceedings of the First Workshop on Data Management for Sensor Networks*. August 2004. (Cited on pages 48 and 49.)

[57] P. Juang, H. Oki, Y. Wang, M. Martonosi, *et al.* Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet. In *Proceedings of ASPLOS X*, pages 96–107. ACM Press New York, NY, USA, 2002. (Cited on page 23.)

[58] C. D. Kidd, R. Orr, G. D. Abowd, C. G. Atkeson, *et al.* The Aware Home: A Living Laboratory for Ubiquitous Computing Research. In *Proceedings of the Second International Workshop on Cooperative Buildings*, pages 191–198. 1999. (Cited on page 33.)

[59] D. E. Krebs. Interpretation standards in locomotor studies. In Rebecca Craik and Carol Oatis, eds., *Gait Analysis: Theory and Application*. Mosby, 1995. (Cited on page 74.)

[60] D. E. Krebs, J. I. Huddleston, D. Goldvasser, D. M. Scarborough, *et al.* 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. (Cited on page 124.)

[61] M. Laibowitz, J. Gips, R. Aylward, A. Pentland, *et al.* A sensor network for social dynamics. In *Proceedings of IPSN '06*, pages 483–491. 19-21 April 2006. (Cited on page 57.)

[62] M. Laibowitz and J. A. Paradiso. Wireless wearable transceivers. *Circuit Cellar*, pages 28–39, February 2004. (Cited on page 57.)

[63] L. Lee. *Gait Analysis for Classification*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 2002. (Cited on page 78.)

[64] J. Lester, T. Choudhury, and G. Borriello. A practical approach to recognizing physical activities. In *Proceedings of Pervasive '06*, pages 1–16. 2006. (Cited on pages 51 and 78.)

[65] R. Li and G. Belford. Instability of decision tree classification algorithms. In *Proceedings of SIGKDD '02*, pages 570–575. 2002. (Cited on page 106.)

[66] J. Lifton, M. Feldmeier, Y. Ono, C. Lewis, *et al.* A platform for ubiquitous sensor deployment in occupational and domestic environments. In *Proceedings of SPOTS '07*. 2007. To appear. (Cited on page 20.)

[67] 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. 19-21 April 2006. (Cited on pages 48 and 49.)

[68] D. Lovell. *A System for Real-Time Gesture Recognition and Classification of Coordinated Motion*. MEng thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, February 2005. (Cited on page 180.)

[69] C. Luschi, M. Sandel, P. Strauch, J. J. Wu, *et al.* Advanced signal-processing algorithms for energy-efficient wireless communications. *Proceedings of the IEEE*, 88(10):1633–1650, October 2000. (Cited on pages 21 and 55.)

[70] Magellan Navigation, Inc. eXplorist 400 North America. http://www.magellangps.com/products/product.asp?PRODID=1071&SEGID=356&tab=0. (Cited on page 24.)

[71] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, *et al.* Wireless sensor networks for habitat monitoring. In *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications*, pages 88–97. September 2002. (Cited on pages 23, 53 and 166.)

[72] M. Malinowski. *CargoNet: Micropower Sensate Tags for Supply-Chain Management and Security*. MEng thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, February 2007. (Cited on page 169.)

[73] D Marsh, R. Tynan, D. O'Kane, and G. M. P. O'Hare. Autonomic wireless sensor networks. *Engineering Applications of Artificial Intelligence*, 17:741–748, 2004. (Cited on page 52.)

[74] D. McIntire, K. Ho, B. Yip, A. Singh, *et al.* The low power energy aware processing (LEAP) embedded networked sensor system. In *Proceedings of IPSN '06*, pages 449–457. 19-21 April 2006. (Cited on page 52.)

[75] D. Merrill and J. A. Paradiso. Personalization, expressivity, and learnability of an implicit mapping strategy for physical interfaces. In *Proceedings of CHI '05, Extended Abstracts*, pages 2152–2161. 2005. (Cited on page 178.)

[76] 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. (Cited on page 81.)

[77] J. Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3(4):319–342, March 1989. (Cited on page 90.)

[78] B. Morgan. *Learning Commonsense Human-language Descriptions from Temporal and Spatial Sensor-network Data*. SM thesis, Program in Media Arts and Sciences, Massachusetts Institute of Technology, September 2006. (Cited on page 161.)

[79] S. J. Morris. *A Shoe-Integrated Sensor System for Wireless Gait Analysis and Real-TimeTherapeutic Feedback*. ScD thesis, Division of Health Sciences and Technology, Massachusetts Institute of Technology, June 2004. (Cited on pages 19, 23, 27, 34, 63, 65, 70 and 123.)

[80] moteiv corporation. http://www.moteiv.com/. (Cited on page 68.)

[81] E. Munguia Tapia. *Activity Recognition in the Home Setting Using Simple and Ubiquitous Sensors*. SM thesis, Program in Media Arts and Sciences, Massachusetts Institute of Technology, September 2003. (Cited on page 33.)

[82] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI repository of machine learning databases, 1998. (Cited on page 104.)

[83] Nokia Corporation. Nokia 5500 sport. http://www.nokia.com/link?cid=PLAIN_TEXT_32998. (Cited on page 24.)

[84] S. W. Norton. Generating better decision trees. In *Proceedings of the Eleventh International Conference on Artificial Intelligence*, pages 800–805. 1989. (Cited on page 93.)

[85] M. Núñez. The use of background knowledge in decision tree induction. *Machine Learning*, 6(3):231–250, May 1991. (Cited on page 93.)

[86] M. Ouwerkerk, F. Pasveer, and N. Engin. SAND: a modular application development platform for miniature wireless sensors. In *Proceedings of BSN '06*. 3-5 April 2006. (Cited on page 68.)

[87] C. Pachetti, F. Mangini, F. Aglieri, C. Fundaro, *et al.* Active music therapy in Parkinson's disease: An integrative method for motor and emotional rehabilitation. *Psychosomatic Medicine*, 62:386–393, 2000. (Cited on page 28.)

[88] J. A. Paradiso and N. Gershenfeld. Musical applications of electric field sensing. *Computer Music Journal*, 21(2):69–89, 1997. (Cited on page 179.)

[89] J. A. Paradiso, K. Hsiao, A. Y. Benbasat, and Z. Teegarden. Design and implementation of expressive footwear. *IBM Systems Journal*, 39(3&4):511–529, 2000. (Cited on pages 57 and 180.)

[90] J. A. Paradiso, S. J. Morris, A. Y. Benbasat, and E. Asmussen. Interactive therapy with instrumented footwear. In *CHI '04 extended abstracts*, pages 1341–1343. ACM Press, New York, NY, USA, 2004. (Cited on page 28.)

[91] A. Pentland. Healthwear: medical technology becomes wearable. *Computer*, 37(5):42–49, May 2004. (Cited on page 24.)

[92] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of Dynamic Voltage Scaling algorithms. In *Proceedings of ISLPED '98*, pages 76–81. 1998. (Cited on pages 48, 53 and 56.)

[93] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Proceedings of IPSN '05*, page 48. 2005. (Cited on page 68.)

[94] J. Polastre, R. Szewczyk, C. Sharp, and D. Culler. The mote revolution: Low power wireless sensor network devices. In *Proceedings of Hot Chips 16: A Symposium on High Performance Chips*. August 2004. (Cited on page 27.)

[95] R. Powers. Batteries for low power electronics. *Proceedings of the IEEE*, pages 687–693, April 1995. (Cited on page 20.)

[96] S. S. Pradhan, J. Kusuma, and K. Ramchandran. Distributed compression in a dense microsensor network. *IEEE Signal Processing Magazine*, 19(2):51–60, March 2002. (Cited on page 167.)

[97] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986. (Cited on page 86.)

[98] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1992. (Cited on pages 89 and 95.)

[99] L. Rabiner. A tutorial on Hidden Markov Models and selected applications in speechrecognition. *Proceedings of the IEEE*, 77:257–86, 1989. (Cited on pages 45 and 83.)

[100] 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. (Cited on pages 22 and 48.)

[101] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava. Energy-aware wireless microsensor networks. *IEEE Signal Processing Magazine*, 19(2):40–50, March 2002. (Cited on page 48.)

[102] M. Rahimi, R. Pon, W. J. Kaiser, G. S. Sukhatme, *et al.* Adaptive sampling for environmental robotics. In *Proceedings of ICRA '04*, volume 4, pages 3537–3544. Apr 26-May 1, 2004. (Cited on pages 19, 48 and 49.)

[103] T. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall, 1996. (Cited on page 55.)

[104] Renata SA. CR1225 technical data sheet. http://www.renata.com/pdf/3vlithium/DBCR1225.03.pdf. (Cited on page 69.)

[105] L. Rokach and O. Maimon. Top-down induction of decision trees classifiers - a survey. *IEEE Transactions on Systems, Man and Cybernetics, Part C*, 35(4):476–487, Nov. 2005. (Cited on pages 86 and 95.)

[106] D. Sachs. *A Forearm Controller and Tactile Display*. SM thesis, Program in Media Arts and Sciences, Massachusetts Institute of Technology, June 2005. (Cited on page 180.)

[107] B. Schott, M. Bajura, J. Czarnaski, J. Flidr, *et al.* A modular power-aware microsensor with >1000x dynamic power range. In *Proceedings of IPSN '05*, pages 469–474. 15 April 2005. (Cited on page 68.)

[108] C. E. Shannon and W. Weaver. *Mathematical Theory of Communication*. University of Illinois Press, 1963. (Cited on page 93.)

[109] E. Shih, P. Bahl, and M. J. Sinclair. Wake on wireless:: an event driven energy saving strategy for battery operated devices. In *Proceedings of MobiCom '02*, pages 160–171. ACM Press New York, NY, USA, 2002. (Cited on page 24.)

[110] E. Shih, S-H. Cho, N. Ickes, R. Min, *et al.* Physical layer driven protocol and algorithm design for energy-efficientwireless sensor networks. In *Proceedings of MobiCom '01*, pages 272 – 287. July 2001. (Cited on page 53.)

[111] A. Sinha, A. Wang, and A. Chandrakasan. Algorithmic transforms for efficient energy scalable computation. In *Proceedings of ISLPED '00*, pages 31–36. 2000. (Cited on pages 21 and 54.)

[112] M. B. Srivastava. Sensor node platforms & energy issues. MobiComp 2002 Tutorial Notes. (Cited on pages 22 and 48.)

[113] T. Starner and J. A. Paradiso. Human generated power for mobile electronics. In C. Piguet, ed., *Low-Power Electronics*, chapter 45. CRC Press, 2004. (Cited on page 34.)

[114] W. D. Stiehl, J. Lieberman, C. Breazeal, L. Basel, *et al.* Design of a therapeutic robotic companion for relational, affective touch. In *IEEE International Workshop on Robot and Human Interactive Communication '05*, pages 408–415. 13-15 Aug. 2005. (Cited on page 71.)

[115] T. Szepesi and K. Shum. Optimizing cellular phone power management for smallest footprint and lowest cost. In *Proceedings of the Communications Design Conference*. 2001. (Cited on page 55.)

[116] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, *et al.* An analysis of a large scale habitat monitoring application. In *Proceedings of SenSys '04*, pages 214–226. November 2004. (Cited on page 23.)

[117] M. Tan. Cost-sensitive learning of classification knowledge and its applications in robotics. *Machine Learning*, 13(1):7–33, October 1993. (Cited on page 93.)

[118] Texas Instruments Inc. MSP430F161x Specifications. http://focus.ti.com/docs/prod/folders/print/msp430f1610.html. (Cited on pages 39 and 128.)

[119] M. H. Thaut, G. P. Kenyon, M. L. Schauer, and G. C. McIntosh. The connection between rhythmicity and brain function. *Proceedings of IEEE Engineering in Medicine and Biology*, pages 101–107, March 1999. (Cited on page 28.)

[120] G. Tolle, D. Gay, W. Hong, J. Polastre, *et al.* A macroscope in the redwoods. In *Proceedings of SenSys '05*, pages 51–63. ACM Press New York, NY, USA, 2005. (Cited on page 20.)

[121] P. Turney. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, 2:369–409, 1995. (Cited on pages 81 and 104.)

[122] P. Turney. Types of cost in inductive concept learning. In *Workshop on Cost-Sensitive Learning at the Seventeenth International Conference on Machine Learning*, pages 15–21. 2000. (Cited on page 84.)

[123] A. Webb. *Statistical Pattern Recognition.* Wiley, New York, 2nd edition, 2002. (Cited on pages 81, 82, 94 and 104.)

[124] L. Welti-Santos. Use of ResEnv Stack by children to control programming animation, music, and light with body movement. Unpublished technical report, July 2003. (Cited on page 177.)

[125] D. A. Winter. Kinematic and kinetic patterns in human gait: Variability and compensating effects. *Human Movement Science*, 3(1-2):51–76, 1984. (Cited on page 125.)

[126] G. Wu and Z. Ladin. The study of kinematic transients in locomotion using the integrated kinematic sensor. *IEEE Transactions on Rehabilitation Engineering*, 4(3):193–200, Sept. 1996. (Cited on page 123.)

[127] L. Yu and A. Ephremides. Detection performance and energy efficiency of sequential detection in a sensor network. In *Proceedings of HICSS '06*, volume 9. Jan. 2006. (Cited on pages 49 and 50.)

[128] C. B. Yun, J. J. Lee, S. K. Kim, and J. W. Kim. Recent R & D activities on structural health monitoring for civil infra-structures in Korea. *KSCE Journal of Civil Engineering*, 7(6):637–51, 2003. (Cited on page 32.)

[129] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi. Hardware design experiences in ZebraNet. In *Proceedings of SenSys '04*, pages 227–238. ACM Press New York, NY, USA, 2004. (Cited on page 23.)

[130] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration. *IEEE Signal Processing Magazine*, March 2002. (Cited on pages 49, 50 and 168.)

[131] H. Zhu, J. J. Wertsch, G. F. Harris, J. D. Loftsgaarden, *et al.* Foot pressure distribution during walking and shuffling. *Arch Phys Med Rehabil*, 72:390–397, May 1991. (Cited on page 124.)