E-Gesture: A Collaborative Architecture for Energy-efficient Gesture Recognition with Hand-worn Sensor and Mobile Devices

Taiwoo Park
Computer Science
KAIST
Daejeon, 305-701, Republic of Korea
twpark@nclab.kaist.ac.kr

Jinwon Lee†
Computer Science
KAIST
Daejeon, 305-701, Republic of Korea
jcircle@nclab.kaist.ac.kr

Inseok Hwang
Computer Science
KAIST
Daejeon, 305-701, Republic of Korea
inseok@nclab.kaist.ac.kr

Chungkuk Yoo
Computer Science
KAIST
Daejeon, 305-701, Republic of Korea
ckyoo@nclab.kaist.ac.kr

Lama Nachman
Intel Corporation
Santa Clara, CA 95052 USA
lama.nachman@intel.com

Junehwa Song
Computer Science
KAIST
Daejeon, 305-701, Republic of Korea
junesong@nclab.kaist.ac.kr

Abstract

Gesture is a promising mobile User Interface modality that enables eyes-free interaction without stopping or impeding movement. In this paper, we present the design, implementation, and evaluation of E-Gesture, an energy-efficient gesture recognition system using a hand-worn sensor device and a smartphone. E-gesture employs a novel gesture recognition architecture carefully crafted by studying sporadic occurrence patterns of gestures in continuous sensor data streams and analyzing the energy consumption characteristics of both sensors and smartphones. We developed a *closed-loop collaborative segmentation architecture*, that can (1) be implemented in resource-scarce sensor devices, (2) adaptively turn off power-hungry motion sensors without compromising recognition accuracy, and (3) reduce false segmentations generated from dynamic changes of body movement. We also developed a *mobile gesture classification architecture* for smartphones that enables HMM-based classification models to better fit multiple mobility situations.

Categories and Subject Descriptors

C.3.3 [**Special Purpose and Application Based Systems**]: Real-time and embedded systems

General Terms

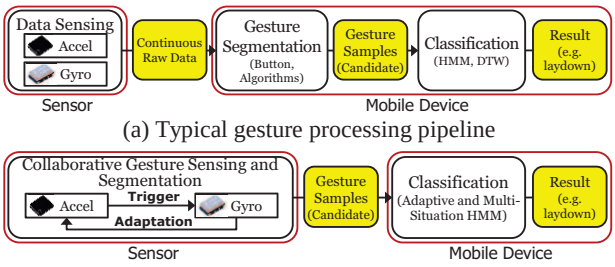
Design, Experimentation, Measurement, Performance

Keywords

Energy Efficiency, Mobile Gesture Recognition, Closed-loop Collaborative Architecture, Hidden Markov Model

1 Introduction

Smartphones have become a constant companion that users keep close by and interact with throughout the day. As a result, users often interact with their smartphones in mobile situations like walking on the street, driving a car, and jogging at a park. Unlike conventional keyboard- and touch screen-based interaction, hand gestures can simplify on-the-move interaction with a phone by reducing the need to take it out and look at it. With the support of a flexible and programmable gesture-based UI, a user can seamlessly interact with various mobile applications without taking her eyes off of the road while driving or slowing her pace while jogging.



(a) Typical gesture processing pipeline

(b) E-Gesture pipeline

Figure 1. Gesture Processing Pipeline

An important challenge in designing a system to support gesture-based mobile UIs is efficient use of the strictly limited energy of mobile and sensor devices. Figure 1(a) represents a typical gesture-processing pipeline for a system consisting of a hand-worn sensor node and a mobile device [12, 21]. The basic processing sequence consists of three stages: (1) collecting and sending sensor data from an accelerometer and/or a gyroscope to the mobile device, (2) detecting and segmenting hand movement samples which may or may not be intended gestures, and (3) classifying the samples as one of several predefined gestures or noise, using methods such as Hidden Markov Model (HMM). A major concern for such a system is that it may quickly drain the batteries of both devices, as they require power-hungry sensors and radio transceivers to be continuously powered. For instance, our measurements of the additional power draw needed to support gesture processing indicate that a smartphone that lasts for 24 hours under normal use would last only 17 hours

SenSys'11, November 1–4, 2011, Seattle, WA, USA.

† Jinwon Lee is now with Qualcomm, Inc.

with a typical gesture-support system.

In addition, gesture processing often becomes more complex as a user moves around. Body movements under different mobile situations incur different characteristics of mobility noise, leading to different sensor-data waveforms for the same hand gestures. Moreover, highly mobile situations like running lead to many non-gestural hand movements such as swinging. Hence, the prevalence of mobility noise and non-gestural movements require careful consideration when designing gesture-based mobile UI systems.

In this paper, we propose *E-Gesture*, a collaborative architecture for energy-efficient gesture recognition that greatly reduces energy consumption while achieving high recognition accuracy[1] under dynamic mobile situations. We observe and characterize users' everyday gestural interactions as well as the effect of gesture recognition on computational load and energy consumption. Our observations led us to redesign the typical mobile gesture-processing pipeline. As shown in Figure 1(b), we designed a multi-level collaborative architecture in which different modules perform given processing tasks in cooperation with each other at two different levels: (1) a sensor node and a mobile device collaboratively perform recognition processing by taking different parts of the task, and (2) the accelerometer and gyroscope collaboratively perform front-end segmentation processing.

1.1 Collaborative Approach to Energy-Efficient Mobile Gesture Recognition

We observe that users mostly perform gestural interactions discretely or sporadically; this leads to intermittent or fragmented bursts of signals instead of continuous, spread ones. Based on these observations, we note that there are several advantages to performing continuous data manipulations such as gesture segmentation at the front end of processing, i.e., in a sensor node. With this approach, a large part of the data stream can be filtered out at an early stage, greatly reducing costly data transfers.

However, developing an efficient sensor-side segmentation architecture presents several practical challenges. First, only lightweight segmentation architectures are practical due to sensor devices' limited computing resources. Second, we discovered that the gyroscope is the most energy-consuming component (i.e., 56%). Controlling the gyroscope to save energy without sacrificing segmentation accuracy is a key challenge. Finally, dynamic mobility situations seriously attenuate the energy-saving benefits of sensor-side segmentation. Elevated noise under mobile situations frequently can cause falsely identified segments, which leads to needlessly powering-on the radio transceiver and gyroscope. For example, a 90% false segmentation rate while a user ran would nullify any energy saved through sensor-side segmentation.

To address these challenges, we propose a *closed-loop collaborative segmentation architecture* on the sensor-side which combines the advantages of two different segmentation methods based on the accelerometer and gyroscope.

[1] If the term 'accuracy' is used without any specification, it refers to the overall gesture recognition accuracy. Note that there are two types of specific accuracy referred in this paper: segmentation accuracy and classification accuracy.

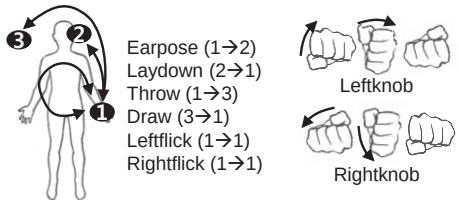


Figure 2. Eight Gestures used for Gesture Collection

Our approach is energy efficient and robust to transitions between common forms of mobility. First, it is lightweight enough to be implemented in a resource-scarce sensor node, as it utilizes a simple but effective threshold-based algorithm. Second, selective activation of the gyroscope provides considerable energy savings and a high level of segmentation accuracy. We observe that the characteristics of the accelerometer-based segmentor are low-energy and low-accuracy (2.47mW and 10-20% errors), whereas those of the gyroscope-based segmentor are high-energy and high-accuracy (26.1mW and 5-10% errors). Thus, we use the gyro-based segmentor only to validate the gesture segments detected by the accel-based segmentor. Finally, our architecture remains energy-efficient in the face of transitions between mobile situations by continuously reconfiguring itself. We observe that the accel-based segmentor is mobility-vulnerable, i.e., error rates of up to 90% while a user is running. In contrast, the gyro-based segmentor is mobility-robust, with errors of 5-10% even in mobile situations. This inspired us to allow the gyro-based segmentor to give feedback to the accel-based segmentor when invalidating false segments. As a result, the accel-based segmentor can be fit to the current mobility situation by regulating the false segment ratio even during mobility situation transitions.

In addition to the energy-efficient architecture, we have extended our gesture classification architecture to effectively deal with different common mobility situations. We developed and compared two HMM architectures: (1) *adaptive* and (2) *multi-situation HMM architecture*. Adaptive HMM adapts to the current mobility situation by continuously updating HMMs. The Multi-situation HMM achieves the best accuracy under known mobility situations by training HMM for each situation in advance.

1.2 Implementation and Evaluation

We have implemented an E-Gesture prototype using a wearable sensor node, i.e., a wrist-watch type sensor node with TinyOS, and off-the-shelf mobile devices i.e., Nokia N96 and Nexus One. We partially ported the widely-used HMM Toolkit, HTK [1], to the smartphones. To the best of our knowledge, ours is the first port of HTK to embedded mobile devices. Also, we have shown the practicality of the E-Gesture prototype by building two gesture-based applications on top of the prototype.

To evaluate the performance of E-Gesture, we carefully select a set of eight intuitive hand gestures for mobile-user interactions, as shown in Figure 2. In addition, we examined four common mobility situations: Standing (*STAND*), Walking (*WALK*), Running (*RUN*), and Riding a car (*RIDE*). Our results show that E-Gesture can improve the energy efficiency of the sensor and the mobile device by up to 2.4 times

and 2.8 times respectively, and classify gestures at an average accuracy of 94.6% under dynamically changing mobility situations. Note that HMMs trained using only the standing situation fail in situations other than standing, providing only 69.78% accuracy in our dynamic environment.

The rest of this paper is organized as follows. We discuss related work in Section 2. Section 3 presents the E-Gesture framework including energy-efficient segmentation and mobility-considered classification architectures. Section 4 describes implementation details, and Section 5 gives the evaluation results. We discuss lessons and considerations in Section 6, and conclude the paper in Section 7.

2 Related work

Gesture recognition has been extensively studied [8, 15] for the last two decades. A rich body of work has focused on vision-based method [7, 26], but these are not suitable for mobile environments due to limited wearability, high energy and computational costs, and sensitivity to light conditions.

Recently, hand-worn accelerometer and gyroscope sensor-based gesture recognition has been successfully used in mobile and pervasive computing. This approach is appealing because it is low cost, low power, requires only lightweight processing, and can be implemented with compact form factors. Several projects have proposed gesture recognition systems based on a wide range of sensing form factors, e.g., Wii Controller [18, 22], Intel PSI board [19, 20], SAMSUNG Pen [3], Microsoft XWand [25], VTT Soapbox [12], GIT Watch [13, 17], and ETH Wearable Computer [8, 23]. These systems have been used for a diverse set of purposes, e.g., controlling mp3 players and other appliances, or writing numbers and characters in the air.

Most existing gesture recognition systems, however, have not prioritized the energy efficiency of sensor and mobile devices in real mobile situations. In typical systems, sensor devices continuously capture hand motions at a high rate (e.g., 100 Hz) and transmit the raw data to a mobile device, expending considerable energy from both devices. With the recent proliferation of smartphones, mobile users often demand long-lasting interactions throughout the day (e.g., controlling their smartphones even while on-the-go). Hence, it is critical to make the energy efficiency of sensors and mobile devices a first-class consideration.

Sensor-side gesture segmentation is essential to extending battery lifetimes. It not only reduces data transmission, but also gives many opportunities to turn off power-hungry motion sensors. E-Gesture employs a threshold-based segmentation method to automatically detect short pauses at the start and end of gestures. Different from complex segmentation methods such as sequence analysis [8] and probability calculation [14], the threshold-based method is lightweight enough to be implemented in a resource-scarce sensor device, e.g., 4KB RAM and 8MHz CPU. However, deploying the threshold-based method in mobile situations has revealed that further engineering is required to deal with the noise caused by body movements. Recently, G. Raffa, et al. [21] developed an efficient architecture for gesture recognition, focusing on early filtering of unwanted data. However, this work does not address the problem of false gesture segmentation due to non-gestural body movements, which can significantly degrade both accuracy and energy efficiency. We carefully designed the E-Gesture architecture to save energy without compromising accuracy even in mobile situations.

Also, automatic gesture segmentation provides a natural user experience. The simplest way to implement sensor-side segmentation is to use a button, i.e., continuously indicate the start and stop of gestures through button push/release [15, 17, 22]. In order to do this, users must wear an external button on their fingers or hold it in their hand. Unlike a wristwatch-like sensor, wearing a button all day is burdensome and un-natural, limiting the usability of a hand gesture system. Furthermore, button-based segmentation does not eliminate the need for automatic gesture segmentation. Additional correction is still required because users have difficulty precisely indicating the beginning and end of gestures.

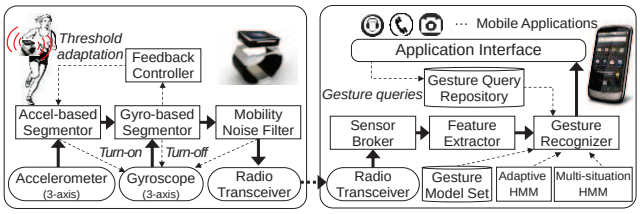
Efficient use of scarce power resources is a longstanding challenge for sensor and mobile systems. Several projects are generally related to E-Gesture in that they carefully trigger sensors to save energy, e.g., a hierarchical sensor management scheme that activates power-hungry sensors only when triggered by power-efficient ones [24], energy-efficient context monitoring frameworks that selectively turn on and off sensors depending on application demands and the current context [9, 10, 11], or an accelerometer-based filter to reduce the power-on time of a power-hungry GPS sensor [6] or a gyroscope [16]. Unlike these systems, E-Gesture is specialized for gesture recognition. Instead of dealing with only low-level sensors, our approach achieves both a high level of segmentation accuracy as well as energy-efficiency. In terms of sensor control schemes, existing systems can be characterized as open-loop sensor controls. Our system, however, employs closed-loop controls, providing continuous feedback via the closed control loop. This ability is essential to rapidly adapt to dynamic situational changes, which are common in mobile environments. E-Gesture continuously reconfigures itself, and, as a result, is both adaptive and robust.

Many gesture classification algorithms have been proposed to classify a segmented gesture into one of many predefined gestures or as garbage, e.g., HMM, Template matching [5, 23], Dynamic Time Wrapping (DTW) [15], and light-weight HMM [2]. Among these, HMM has been widely used since it has shown the best accuracy [3, 8, 12, 14]. A typical HMM-based gesture classifier is trained with gesture samples from stationary situations. Hence, it is prone to misclassifying gesture segments that show different sensor data characteristics in other mobility situations. To address this problem, E-Gesture employs a novel gesture classification architecture to fit HMMs to multiple mobility situations.

3 E-Gesture Architecture Design

The E-Gesture framework facilitates the development and provision of gesture-based UIs for mobile applications, providing APIs and an efficient runtime. The architecture spans over a sensor device and a mobile device, which are a wristwatch-type sensor mote and a smartphone, respectively.

Sensor-side Architecture: the sensor device effectively performs *motion sensing* and *gesture segmentation* as shown



(a) Sensor-side Architecture (b) Mobile device-side Architecture

Figure 3. E-Gesture Architecture

in Figure 3(a). For accurate motion sensing, we use a 3-axis accelerometer and 3-axis gyroscope. While the accelerometer is often used in gesture recognition systems [3, 5, 8], recent research demonstrates that additional use of a gyroscope can improve gesture classification accuracy [21] by up to 4% because it can detect rotational movements. In particular, we develop a novel gesture segmentation architecture for computationally limited sensor devices, which achieves a high level of energy efficiency under dynamically changing mobility situations. For this, we develop the *accelerometer-based and gyroscope-based segmentor*, and *feedback controller* (see Section 3.1). In addition, a *mobility noise filter* removes false segmentation in on-the-go situations.

Mobile device-side Architecture*:* the mobile device effectively performs *sensor brokering*, *feature extraction*, *gesture classification*, and provides an *application interface* as shown in Figure 3(b). The *sensor broker* receives the segmented gesture data over ZigBee or Bluetooth from the sensor and then interprets the data. To achieve accurate classification under dynamic mobility situations, we propose two mobile gesture classification architectures, i.e., *adaptive and multi-situation HMMs* (see Section 3.2). In addition, the *application interface* manages interactions with applications.

3.1 Energy Savings of Sensor-side Segmentation

Sensor-side Savings: We show that sensor-side segmentation saves energy for the sensor device as well as the mobile device. We first look into the energy usage pattern of our sensor device (see Section 4 for details). Figure 4 summarizes the energy usage[2] of each component. The gyroscope consumes the most in the sensor device (56%), using 10 times more energy than the accelerometer (5.3%). Hence, reducing the power-on time of the gyroscope would significantly improve overall energy efficiency. We integrate our gyroscope control scheme into the *closed-loop collaborative segmentation* discussed in Section 3.2.

Figure 4 shows that radio transmission consumes the second most amount of energy (21%). This observation motivates the need for sensor-side segmentation. Since gestures occur sporadically, the radio transceiver should be triggered only after a segment has been detected. According to recent work [21], over the course of a typical user's day only 5% of continuous sensor data would be identified as part of a gesture. Note that other basic activities such as CPU operation and radio listening consume relatively little energy (17%).

[2] Each second, a sensor samples 22 bytes of data from the accelerometer and gyroscope at 40Hz, and transmits via Zigbee (see Section 5.2 for details). Basic consumption includes CPU operation and radio listening (LPL, 4% duty cycle).

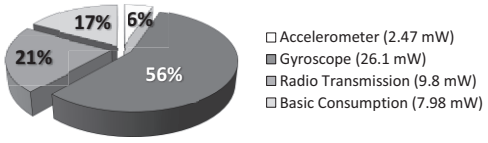


Figure 4. Energy Consumption Profile of Sensor Device

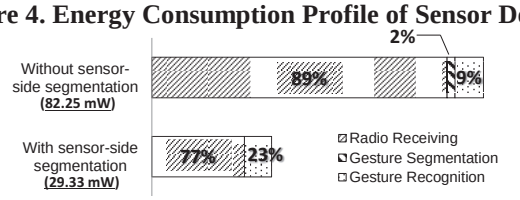


Figure 5. Energy Consumption Profile of Mobile Device

Mobile-device-side Savings: We then look into the energy usage pattern of our mobile device, i.e., the Google Nexus One (see Section 4 for detail). Figure 5 summarizes the energy usage[3] of each major operation. In particular, we measure energy consumption with and without sensor-side segmentation. The main difference is that, without sensor-side segmentation, the mobile device must receive all raw data from the sensor device and extract segments from it. Note that, in both cases, the same sets of gesture segments are processed at the gesture classification stage.

As we can see, sensor-side segmentation yields significant energy savings, i.e., from 82.25 mW to 29.33 mW. With sensor-side segmentation, the radio transceiver not continuously listening for data reduces the energy consumption of the mobile device by 64%. The savings is mainly due to the sleep mode of the Bluetooth radio transceiver. In the Nexus One, sleep mode is automatically triggered if no data is received for a predefined time, i.e., 0.63 sec. More importantly, the infrequent occurrence of gesture segments greatly increases the time spent in sleep mode. Note that sensor-side segmentation hardly affects the energy required for gesture classification by HMM, which is still low compared to radio receiving. In Section 5.2, we discuss the results of experiments measuring the current draws and energy consumption of the mobile device, including the effects of gesture-occurrence frequency.

3.2 Gesture Segmentation in Sensor Device

We propose a *closed-loop collaborative segmentation architecture*, which achieves low-energy consumption and high-accuracy under dynamic mobility situations. As shown in Figure 4, the excessive energy consumption of the gyroscope strongly suggests that reducing the power-on time of the gyroscope is a promising strategy for saving energy in the sensor device. However, energy efficiency must be considered in tandem with accuracy. Our architecture is designed to achieve acceptable accuracy while limiting the energy expended by the sensors.

Furthermore, our architecture addresses several challenges raised in real mobile settings. First of all, the dynamic changes of mobility situations (e.g., a user may sit, run, and

[3] Results are drawn from Nexus One, under dedicated usage without turning on the display, GPS, WiFi, cellular service, etc. The average length of gestures is 1s and the average time gap between gestures is 10s. For gesture classification, HMM based on HTK [1] is used to classify 8 gestures (see Section 5.2).

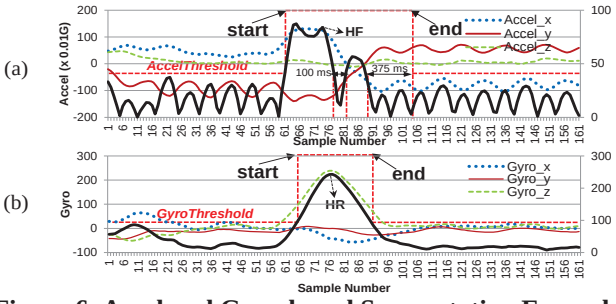


Figure 6. Accel and Gyro-based Segmentation Example

then walk) continuously threaten energy efficiency. Intensive body movements elevate noise levels, frequently generating false segments that unnecessarily consume sensor energy. As a result, our architecture continuously reconfigures itself as the mobility changes, keeping the false segmentation rate within 8%-20% regardless of the mobility situation (see Section 5.4). Finally, sensor devices' resource constraints limit our implementation choices. We employ threshold-based gesture segmentation methods, which are effective but simple enough to be implemented in the sensor device.

3.2.1 Accel- vs. Gyro-based Segmentation

Accel-based Gesture Segmentation. The idea of accel-based gesture segmentation is to approximate the force exerted by a hand (HF) as follows.

$$HF = \sqrt{Accel\_x^2 + Accel\_y^2 + Accel\_z^2} - G(Gravity)$$

The HF is zero if there is no hand motion. The advantage of this approximation is that HF is independent of the wristwatch sensor's orientation; it is difficult to correctly determine the orientation of a sensor device only using the accelerometer. Furthermore, HF can be used to simply model many types of hand gestures [3].

We assume that a gesture is triggered if the HF is greater than a threshold, *AccelThreshold*, i.e., a small constant such as 0.15G for the standing situation. The triggered gesture can be assumed to have stopped if the HF is less than the threshold. However, such a loose stop condition induces *a gesture splitting problem* [3, 5], the unexpected splitting of a single gesture. To avoid gesture splitting, we set an additional temporal threshold. Segments that occur within the temporal threshold of one another are assumed to be part of the same gesture and are merged. A gesture is assumed to have stopped if its duration is greater than the threshold. We empirically determine the value of the temporal threshold, i.e., 375 ms. Figure 6(a) is an example segmentation based on this method for the 'earpose' gesture while running.

Gyro-based Gesture Segmentation The main idea of gyroscope-based gesture segmentation is to approximate the norm of hand rotation (HR) as follows.

$$HR = \sqrt{Gyro\_x^2 + Gyro\_y^2 + Gyro\_z^2}$$

Rotational movement around any axis can increase the HR value. Similar to accel-based HF, gyro-based HR can be used to segment the gesture. We assume that a gesture is triggered if the HR is greater than a *GyroThreshold*, i.e., a small constant such as 25 degree/sec in our empirical study. The triggered gesture is assumed to have ended if the HR is less than the threshold. As opposed to accel-based segmentation, gyro-based segmentation did not produce any split gestures from the samples we collected, as shown in Figure 6(b). Thus, we did not set a temporal threshold for gyroscope-based segmentation.

Segmentation method	Mobility Situation			
	RIDE	STAND	WALK	RUN
Accel-based	0.15G	0.15G	0.2G	0.35G
Gyro-based	25 degree/sec			

Table 1. Optimal Thresholds for Mobility Situations

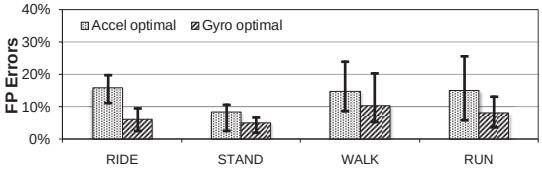


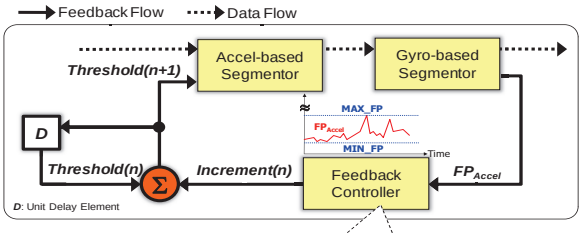
Figure 7. Segmentation Accuracy

Segmentation Accuracy of Accel and Gyro Interestingly, we found that gyro-based segmentation provides higher segmentation accuracy than accel-based segmentation, regardless of the mobility situation. We tested the segmentation accuracy of the two methods based on real continuous motion data containing both gestures and non-gestures. Data collected from 7 participants were tested user-dependently under four mobility situations, i.e., RIDE, STAND, WALK and RUN. Section 5 elaborates on the collected motion data. For each method and each mobility situation, we empirically found a threshold that minimized the false-positives (FPs) without incurring false-negatives (FNs). As shown in Table 1, the thresholds of the gyro-based method were the same regardless of the mobility, whereas thresholds for the accel-based one generally increased with the degree of mobility.

Figure 7 shows differences between the average FPs for two segmentation methods, with bars indicating the minimum and maximum. Gyro-based segmentation achieves a low FP error rate, i.e., less than 8% on average, while accel-based segmentation achieves 15% on average. Note that the threshold of the gyro-based method is fixed for all mobility situations. We conjecture that the difference in segmentation accuracy is mainly a result of the different sensing modalities and capabilities of the sensors. The rotational movements measured by the gyroscope are dominated by hand rotation, rather than body rotation; body rotation is normally less frequent and much slower than hand rotation, and can be easily filtered out by our *GyroThreshold*. In contrast, linear movements measured by an accelerometer are not caused by hand motions alone; gravity and body movements also impact the measured values. As a result, the accel-based method might generate more segmentation errors than the gyro-based. As in Figure 6, several small waves were generated in the accel-based method due to the running movements of the body, but not in the gyro-based method.

3.2.2 Closed-loop Collaborative Segmentation

Figure 8 shows the operational flow of closed-loop collaborative segmentation. The observations above inspired the collaborative segmentation approach. That is, we only use the high-energy, high-accuracy gyro-based segmentor to validate the gesture segments detected by the accel-based



$$\text{If } (FP_{Accel}(n) > MAX\_FP)$$
$$Increment(n) = \alpha(FP_{Accel}(n) - MAX\_FP)$$
$$\text{If } (FP_{Accel}(n) < MIN\_FP)$$
$$Increment(n) = \beta(FP_{Accel}(n) - MIN\_FP)$$
$$Threshold(n+1) = Threshold(n) + Increment(n)$$

where, Threshold(n) and $FP_{Accel}(n)$ denote current threshold and false-positive rate, respectively. α and β are empirically determined small constants.

Figure 8. Overall Processing Flow of Closed-loop Collaborative Segmentation Architecture

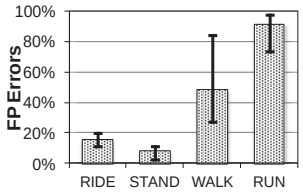


Figure 9. FP with a Low *AccelThreshold*

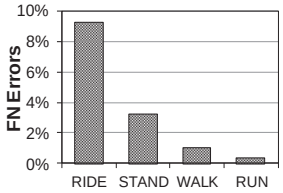


Figure 10. FN with a High *AccelThreshold*

segmentor, thereby achieving low-energy as well as high-accuracy. Moreover, we introduce the closed-loop reconfiguration of accel-based segmentor to continuously adapt to the dynamic changes of mobility situations.

The keys to our closed-loop collaborative segmentation are twofold. First, the accel- and gyro-based segmentors are connected in series, and the latter is only activated when the former detects gesture segments. This reduces the power-on time of the highest-energy gyroscope. Likewise, the radio transceiver is activated when the gyro-based segmentor validates the segments from the accel-based segmentor. Second and more importantly, the gyro-based segmentor adaptively reconfigures the accel-based segmentor under dynamically changing mobility situations, thereby minimizing false segmentation by the accel-based segmentor. Thus, fewer activations of subsequent components like the gyroscope and radio transceiver reduces overall energy consumption.

Specifically, we found that on-the-go situations require a high threshold for the accel-based segmentor to avoid FP segments. Such FP segments, resulting from increased body-movement noise, would waste energy as well as trigger actions unintended by the user. In contrast, stationary situations require a low threshold to avoid FN segments, which users may perceive as unresponsiveness. The following two experiments demonstrate that threshold adaptation is crucial under different mobility situations. First, we measure the FP ratio while fixing the threshold to the lowest value (0.15G) in four mobility situations (see Table 1). As shown in Figure 9, the FP ratio under WALK and RUN increases to 100%. Second, we measure the FN ratio, fixing the threshold to the highest value (0.35G). As in Figure 10, the FN ratio under RIDE and STAND increases to 10%. Note that gyro-based methods show stable accuracy in both types of errors with a constant threshold of 25 degree/s, for all mobility situations.

Runtime Threshold Adaptation Policy. We developed a runtime threshold adaptation policy. Figure 8 illustrates our linear feedback controller implementing this policy. It is simple to implement in resource-limited sensors and responsive to mobility changes. More sophisticated controllers would be inappropriate given sensors' resource constraints.

Our key policy idea is as follows. The gyro-based segmentor monitors FP_{Accel}, denoting the ratio of FPs segmented by accel-based segmentor but invalidated by the gyro-based segmentor. It then regulates FP_{Accel} by adaptively controlling the threshold of the accel-based segmentor. In effect, the threshold is raised if the FP_{Accel} exceeds the maximum tolerable limit (denoted as MAX_FP). On the other hand, the threshold is lowered if FP_{Accel} is below the minimum tolerable limit (denoted as MIN_FP). Our experimental results characterize the tolerable range of FP_{Accel}; a properly configured accel-based segmentor typically yields 5~10% more FPs than those of the gyro-based segmentor.

In our current implementation, we conservatively fix the MIN_FP and MAX_FP to 8% and 20%, respectively, in order to cope with runtime and inter-user variability. To measure FP_{Accel} at the gyro-based segmentor, we set the window size to 20 sec, sliding every 1 sec, which is similar to existing body activity recognition methods [4]. According to our experiments under dynamically changing mobility situations, this implementation achieves a near-zero (0.39%) FN ratio. It also achieves an FP ratio (10%) that is comparable to the optimal threshold (see Section 5.4 for details).

3.2.3 Mobility Noise Filter

While closed-loop collaborative segmentation mostly removed false gesture segments caused by dynamic mobility changes, some false gesture segments still occurred, especially in on-the-go situations. Notably, a "hand swing" gesture was the most frequent false gesture since people naturally swing their hands while walking or running. In response, we developed a simple but effective filtering heuristic, i.e., *discard gestures longer than 2 sec*. We observe that hand swings are often much longer than legitimate gestures. By default, we immediately start data transmission upon detecting a gesture segment. But the radio transceiver and the gyroscope are turned-off after a timeout to avoid false gestures caused by non-gestural hand swings. Our experiments show that in a RUN situation, this noise filter reduces radio transmission energy by 28% and 20% of gyroscope energy (see Section 5.2).

3.3 Gesture Classification in Mobile Device

Accuracy is a primary consideration when performing gesture classification on a mobile device. However, mobility noise caused by body movements seriously degrades accuracy due to changes in sensor data characteristics. Moreover, noise levels frequently change over time, which reduces the effectiveness of conventional approaches that build and train an HMM model per gesture only in a stationary situation. In our experiments, the accuracy of an HMM trained for STAND drops from 98.4% to 53.3% under RUN.

To address this challenge, we propose two mobile gesture classification architectures, i.e., *adaptive* and *multi-situation HMMs*, which can accurately classify gestures under dynamic mobility situations. In this work, we do not aim to explore the most effective mobile gesture classification architecture, but to determine whether our proposed architectures can improve classification accuracy under dynamic mobility situations.

3.3.1 Basic Gesture Classification Architecture

The core of our gesture classification is HMM [1]. For each gesture, an HMM model is built and trained based on a set of collected gesture samples. We train the model using the standard Baum-Welch re-estimation algorithm. We also build a garbage HMM model to filter out non-gestural movements or undefined gestures that frequently occur, e.g., 'raising a hand to eat food'. As commonly used in other gesture recognition works[2, 8], ergodic topology [14] is used to derive a garbage model using trained gesture models.

At runtime, a potential gesture segmented by the sensor device is classified as one of the defined gestures or garbage. We calculate a Feature Vector (*FV*) series containing 18 elements, i.e., raw data, delta data, and integral data for each axis of the 3D accelerometer and the 3D gyroscope from the segmented samples. From the FV series, we use the Viterbi algorithm [1] to efficiently calculate the likelihood of each gesture model. Then, the gesture model *M* with the highest likelihood is selected as the classified gesture. However, if the garbage model is picked up, we reject the gesture. In our current implementation, we configure HMM using common gesture classification techniques [12, 17, 21], i.e., 8-state left-right HMM model, Continuous Density HMM (CHMM) with one Gaussian mixture without data quantization.

3.3.2 Adaptive vs. Multi-situation HMM

Based on basic HMM, we propose two mobile gesture classification architectures (see Figure 11). We discuss their advantages and disadvantages later in the Discussion Section.

Adaptive HMM Architecture. The idea of an adaptive HMM architecture is to update the HMM models continuously to better fit the models to the current mobility situation. For runtime HMM adaptations, we employ supervised *Maximum Likelihood Linear Regression* (*MLLR*) [1] which has been developed and widely used for speaker adaptation in speech recognition. Based on a small amount of adaptation data, MLLR computes a set of linear transformations that reduce the mismatch between an initial model set (including a garbage model) and the adaptation data. It is implemented based on *HERest.exe* in the HMM Toolkit [1]. To acquire gesture adaptation data, we borrow the negative update scheme from previous research [15]. In this scheme, adaptation data is obtained whenever a gesture is incorrectly classified during system use, which potentially means a transition to another mobility situation. Upon detecting a classification error, the user notifies the system and provides the correct gesture by pressing a corresponding button.

Multi-situation HMM Architecture. The idea of the multi-situation HMM architecture is to build HMM models for each mobility situation. Since multi-situation HMMs

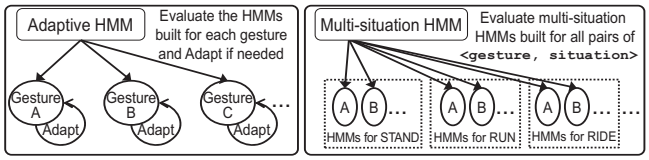


Figure 11. Gesture Classification Architectures

Figure 12. Prototype Implementation

reflect the unique characteristics of gesture data in each mobility situation, they capture gestures performed under different mobility situations. To train the multi-situation HMMs, a sufficient amount of gesture data per situation needs to be collected offline. The garbage model is derived from all models in the same way as the basic model. At runtime, the multi-situation HMM models are decoded at once by the Viterbi algorithm. Among all HMM models, the Viterbi algorithm finds the gesture model under a specific mobility situation that best-fits a given gesture segment. In greater detail, a HMM model is built and trained for each pair of `<gesture, mobility situation>`. If the number of defined gestures is *M* and the number of mobility situations is *N*, we build and train *M* * *N* HMM models. Given a gesture segment, a "Viterbi score" is calculated for each gesture model and each mobility situation, and the best candidate is selected. Based on our preliminary observations, we assume that STAND, WALK, RUN, and RIDE are representative mobility situations. Thus, we collect gesture samples in four different mobility situations and train four HMM models for each gesture.

4 Implementation

4.1 E-Gesture Prototype

As shown in Figure 12, we have implemented an E-Gesture prototype with a smartphone and a wristwatch-style sensor device. The prototype has been deployed on two smartphones: a Nokia N96 (Dual ARM 264MHz CPU and 128MB RAM) and a Google Nexus One (QSD 8250 1GHz CPU and 512MB RAM). For the Nokia N96, we used S60 3.2.0 SDK with TRK debugger. We used the Symbian C++ library to develop the Bluetooth module and other miscellaneous parts, as well as the Open C++ library to develop HMM-based classification modules. For the Google Nexus One, we used Android SDK 2.1 and NDK rev.3.

We used a sensor mote with many sensors, including a 3-axis accelerometer (i.e., ADXL335) and a 3-axis gyroscope (i.e., 3 XV-3500CB) for gesture recognition. The Atmega128L processor is equipped with 4KB RAM and operates at 7.3728 MHz. In addition, the mote supports Bluetooth and ZigBee, a vibrator for tactile feedback, and a ring button for user annotation. Currently, we sample the accelerometer and the gyroscope at a rate of 40Hz. The data is

Class	E-Gesture APIs
Gesture Set Browsing	`browseGestureSet()`
Gesture Set Customization	`addGesture(Name, TrainSamples[])` `deleteGesture(GestureName)`
Gesture Query	`QID = registerQuery(AppName, GestureName,CallbackListener)` `deregisterQuery(QID)`

Gesture Property	`Gesture.power(), Gesture.numOfRepeat()`
	`Gesture.angle(), Gesture.distance()`
	`Gesture.startTime(), Gesture.endTime()`

Table 2. E-Gesture APIs

transmitted over ZigBee since it was impossible to control the on/off of our Bluetooth chipset, which automatically goes into power-save mode after 4 secs without transmission. Due to the absence of a Zigbee chipset on the smartphones, we attached a bridge node to the smartphones for Zigbee-Bluetooth conversion. The bridge node consumes 31.35 mW under the same setting for gesture data transmission as noted in Figure 5. The sensor platform was developed in NesC on TinyOS 1.1.15 and consists of about 1,300 lines of code.

Latency of hardware components. One potential concern for our segmentation architecture is the latency of turning on the gyroscope and the radio transceiver at runtime. The turn-on latency of the gyroscope causes a few gesture samples to be dropped, degrading the recognition accuracy. Most widely used gyroscopes, e.g., Analog Devices ADXRS620 or Invensense IDG-500, typically take 50ms for startup. At a sampling rate of 40Hz, this latency causes two samples to be dropped. In our tests, such drops led to accuracy losses of only 0.4% or less. We reduced the latency by carefully examining alternate gyroscopes and using Epson XV-3500CB, which takes less than 30ms for startup. This allowed us to save one more sample from being dropped, and provided accuracy that was equivalent to not controlling the gyroscope's power at all.

The latency of the radio transceiver is also important since it dictates an application's responsiveness to user interactions. The radio transceiver in the sensor node, i.e., CC2420, achieves less than 0.6ms startup time, meaning that its latency is negligible; a single gesture interaction mostly finishes within 1 to 2 seconds.

HTK Porting to Mobile Device. The key contribution of our implementation is porting the HMM Toolkit (HTK) [1] to a smartphone. To the best of our knowledge, we are the first to have performed this port, and we intend to make it available to other HTK communities. The limited memories of smartphones, especially the Nokia N96, was a major challenge. To address this problem, we extended the default stack size to 80KB and allocated most big stack variables on the large heap memory (16MB). Finally, we confirmed successful working of HVite module which is responsible for classification in HTK. We currently train the HMM parameters on a desktop and then copy the model files, i.e., commands, dict, macros, and lattice, to the smartphones. At runtime, a smartphone generates an input file based on gesture segments received from the sensor device, and passes this file to HVite module for gesture classification.

Figure 13. Swan Boat

Application Interface. E-Gesture APIs allow application developers to specify gesture-based UIs easily without having to be concerned with low-level gesture recognition details such as energy-efficiency and accuracy under dynamic mobility situations. Table 2 shows the set of APIs currently supported by E-Gesture. *registerQuery()* is a key API which specifies both *a gesture of interest* and *a callback function* in the application, which is called when the registered gesture is detected. In addition, we provide gesture property APIs to easily derive the properties of recognized gestures.

4.2 Prototype Application and Field Experience

Using our E-Gesture prototype, we have developed two gesture-based applications: Swan Boat and Mobile mp3 player. By using E-Gesture, we could easily define and quickly implement application-specific gesture UIs to take advantage of our energy- and accuracy-aware architecture.

Swan Boat is a collaborative exergame using hand gestures and treadmill running as its major game inputs. It is a team-race in which pairs of runners sail a virtual boat, closely collaborating to maneuver their boat along curved courses. The boat is steered by the pair's pace differential. Gestural interactions make Swan Boat more exciting and competitive, as shown in Figure 12. Players can attack other teams' boats using punching gestures, and can make their boat skim the surface of the water by flapping their hands.

The first Swan Boat prototype was implemented with naïve gesture processing components, without considering energy and mobility issues. User studies revealed major problems with the game's gesture recognition engine such as (1) rapid sensor battery depletion, and (2) frequent misclassification of gestures. The batteries of the sensor nodes had to be recharged every day, and users frequently complained of missed gesture inputs. We identified that the major causes of poor gesture recognition were that treadmill running generated a substantial amount of body movement noise. Even worse, players continuously changed their running pace or even stopped during play, which led to frequent changes in the noise characteristics. Misclassified or ignored gestures and even unintended FP gestures frequently damaged users' gaming experience. As one player stated: "*I almost gave up attacking the opponents because my punches didn't work. Rather I tried to get faster than others to win the game.*"

We re-engineered Swan Boat by employing E-Gesture. As a result, players and operators reported that (1) the battery recharge cycle was extended by more than two times and (2) misclassifications and FPs were reduced by more than half while ignored cases were nearly eliminated. This feedback is consistent with our experimental results in Section 5, which show that E-Gesture dramatically reduces errors in high mobility situations.

Mobile mp3 player is an mp3 player featuring eye- and touch-free controls even while the user is walking or running (see Figure 12). While listening to music from a Bluetooth headset connected to the smartphone, a user can control the mp3 player through a hand-worn sensor node. Unlike conventional touch-based applications, our mobile mp3 player features intuitive gesture commands as shown in Figure 2. For example, 'left/right knob' adjusts the volume, 'left/right flick' skips forward/backward in the play list, and 'ear-pose/laydown' answers an incoming call (and pauses the music) and hangs up (and resumes the music). All these commands are eye-free, ensuring seamless running without endangering the user by taking her eyes off the road. In addition, tactile feedback is used to acknowledge of these actions and enrich a user's experience.

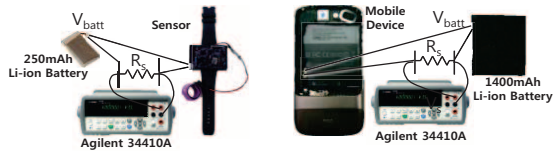
5 Performance Evaluation

We have conducted extensive experiments to evaluate E-Gesture. We demonstrate its energy efficiency in each mobility situation, as well as its overall efficiency under a realistic mixed-mobility usage pattern. We also analyze the effects of dynamic threshold adaptation, which suppresses FPs and FNs in a nearly mobility-independent fashion. Finally, we test the classification accuracy of adaptive and multi-situation HMMs in dynamic mobility situations.

5.1 Gesture Collection

As previously mentioned, we used a set of eight intuitive hand gestures. Figure 2 shows the gesture names and their descriptions, which we used to instruct participants during gesture collection. Next, we examined four common mobility situations: Standing (*STAND*), Walking (*WALK*), Running (*RUN*), Riding a car (*RIDE*). During the *STAND* situation, we requested that users stand upright while performing gestures. During the *WALK* situation, users walk continuously on a treadmill at about 3 mph. During the *RUN* situation, users run at about 5 to 7 mph on a treadmill. During the *RIDE* situation, users perform gestures while riding in the passenger seat of a compact hatchback, i.e., a KIA Picanto. Interestingly, we observed that mobility noise during RIDE situations is the lowest, which is quite different from our expectations. While sensor noise caused by car movements was negligible, sitting posture allows users to gesture more stably, similarly to the STAND situation.

We collected gesture samples of the aforementioned specification from 7 participants, each of whom was a male graduate student ranging in age from 24 to 36. For each mobility situation, each participant performed the set of gestures 30 times. As a result, we have collected 8 gestures × 30 times × 4 situations × 7 participants = 6720 gesture samples in total. Participants wore the sensor node on their left wrist and ring button on their left index finger. We recommended that the participants complete each single gesture within 5 sec. We also asked participants to push and release the button on their left index finger to annotate each gesture segment, providing ground truth for our later analysis. We collected samples over 45 hours in total, spanning 6 days within a period of three weeks.



(a) Sensor-side (b) Mobile-side

Figure 14. Energy Measurement Setup

In addition to gesture samples, we also logged "non-gesture samples" which were collected during "waiting" periods between gestures. The test workload consisting of non-gesture and gesture samples was then used to replay realistic, sporadic user behavior with a gesture-based mobile application. In our collected data, gesture and non-gesture samples accounted for 40% and 60% of the collected data, respectively. Based on the collected data, we synthesized test workloads with a desired gesture ratio by properly mixing gesture samples and non-gesture ones; the default ratio of gesture sample in the workload is configured to 10% in the following experiments.

5.2 Overall Energy-efficiency of E-Gesture

In this subsection, we demonstrate the energy-efficiency of E-Gesture under a dynamic mobility situation. For the experiment, we use the wearable sensor device and Nexus One smartphone as described in Section 4.

5.2.1 Sensor-side Energy Efficiency

Energy Profiling: Figure 14(a) illustrates our measurement setup for energy profiling the sensor device. The sensor node is powered by a coin-size Li-Ion battery (3.7V, 250 mAh) [16]. Due to the compact form factor of the wearable sensor device, the battery capacity of a sensor device is commonly smaller than that of a mobile device, e.g., 1400 mAh in our setting. We measure the voltages (V_s) across a resister (5Ω R_s) in real-time with a digital multimeter. The current consumed by the sensor device can be calculated by dividing the V_s by R_s. Then, the instantaneous power consumption is derived as follows.

$$P_s(t) = (V_{batt} - V_s) \times (V_s(t)/ R_s)$$

The energy consumption is finally obtained as follows.

$$E_s(t) = \sum_{i=0}^{n-1} P_s(t_i) \times (t_{i+1} - t_i)$$

Table 3 shows the profiled results of per-component energy consumption for the gyroscope, accelerometer, radio transceiver, as well as the basic consumption.

Alternatives and Metrics: We compare the energy consumption of three different sensor-side architectures: (1) *no sensor-side segmentor* that continuously samples and sends raw acceleration and gyroscope data to the mobile device, (2) *naïve sensor-side segmentor* that continuously samples raw acceleration and gyroscope data, but sends only the segmented data to the mobile device through sensor-side segmentation, and finally, (3) the closed-loop sensor-side segmentor used in E-Gesture, which adaptively turns on the gyroscope only when needed for segmentation and sends only the segmented data to mobile device. Based on our measurements, we also estimate the battery lifetime under the condition of no processing other than gesture processing. Since no prior work explicitly proposed *automatic* and *sen-*

	Energy Consumption	Battery Lifetime
No sensor-side segmentor	46 mW	20 hrs
Naïve sensor-side segmentor	39 mW	23.7 hrs
Closed-loop segmentor	19 mW	48.7 hrs

Table 3. Sensor-side Energy Efficiency Measurement

	Energy Consumption (mW)			Battery Lifetime (hrs)
	Data Receiving	Segment-ation	Running HMM	
Without sensor-side segmentation	82.25			42.1*
	73.34	1.95	6.96	
With sensor-side segmentation	29.33			74*
	22.68	-	6.65	

*: Under assumption that 3G and WiFi are turned on

Table 4. Mobile-side Energy Efficiency Measurement

sor-side gesture segmentation in different *mobility situation*s, it is difficult to find alternatives for direct comparison. Nonetheless, we carefully suggest that the previous works mostly fall into the category of (1) or (2).

Sensor Data Workload: Due to the immobility of our setup as shown in Figure 14(a), it is impractical to measure the real-time energy consumption of the sensor device while it is actively used in mobile situations. Instead, we used the workloads collected from sensors used under real mobility situations, and computed their energy consumption based on the per-component energy consumption we profiled. To ensure a realistic workload reflecting a user who changes her mobility situation from time to time, we assume a user who uses E-Gesture applications for eight hours a day, including four hours of STAND (covering mostly stationary situations including standing and sitting), two hours of RIDE, and one hour of WALK and RUN, respectively. Then, without loss of generality, we constructed scaled-down 20-min versions of workloads while preserving the above ratios of each situation. The workloads consist of both gesture and non-gesture data collected in Section 5.1.

Experimental results: Table 3 shows the results of our experiments. Our closed-loop segmentor provides significant energy savings for the sensor device. It reduces energy consumption by 58.7%, providing 2.4 times longer battery lifetime compared to approaches without a sensor-side segmentor. Interestingly, the naïve segmentor reduces energy consumption by only 15.2%. This is mainly because the gyroscope, which is the most energy-consuming component, runs continuously in the naïve case.

5.2.2 Mobile Device-side Energy Efficiency

Energy Measurement: Figure 14(b) shows our measurement setup to profile the energy consumed by the mobile device. This is similar to the setup for the sensor device. Our test bed is a Google Nexus One powered by a Li-Ion battery (3.7 V, 1400 mAh), running Android OS 2.2. To exactly measure the energy consumed for gesture processing, we turned off irrelevant services and components including GPS, WiFi, cellular services, and the display. We also shut down all other applications. The battery voltage (V_{batt}) is measured with the Battery Indicator Application. To generate a real-time sensor data workload, we implement a simple data sender that feeds the workload synthesized in Section 5.1.

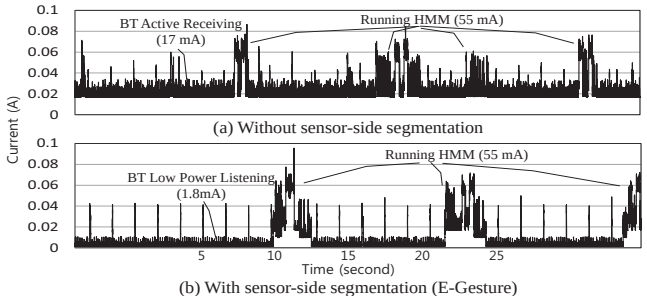


Figure 15. Real-time Energy Consumption in Mobile Device

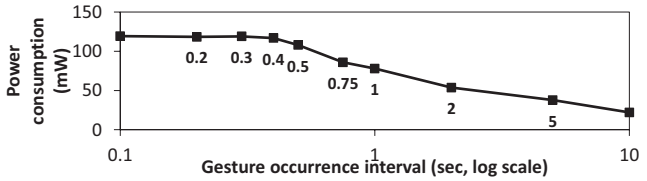


Figure 16. Energy Consumption with Gesture Interval

Alternative: We compare the energy consumption of two different mobile device-side architectures: with and without *sensor-side segmentation*, which represent E-Gesture and existing HMM-based gesture processing systems, respectively. In the architecture with sensor-side segmentation, the mobile device needs to receive only the segmented data and classify gestures by running HMMs. In the alternative without sensor-side segmentation, the mobile device continuously receives raw data from the sensor device, segments data, and classifies gestures by HMM.

Experimental results: Figure 15 shows the run-time energy consumed by our Google Nexus One for 30 seconds. The mobile-side architecture without sensor-side segmentation shows much higher energy consumption. This is mainly because the rapidly incoming raw data prevents the Bluetooth module of the mobile device from entering sleep mode. The current draw of the device when actively receiving data is 17 mA, whereas in sleep mode it only draws 1.8 mA. In both architectures, the computations with HMM rapidly raise the current up to 55 mA, as they actively drive the CPU. However, running the HMM occurs sporadically while receiving data over Bluetooth is continuous. Hence, the energy consumption of the Bluetooth dominates the overall energy consumption of mobile device.

Table 4 summarizes the results. As expected, sensor-side segmentation reduces the energy consumed by the mobile device by 64.3%. It is worth noting that the savings is still 26.2% if we take into account the energy consumed by the Zigbee-Bluetooth bridge node. Table 4 also shows a more detailed breakdown of the total energy consumption of the mobile device. We see that receiving data over Bluetooth dominates the energy consumption of the mobile device, and that sensor-side segmentation is essential for saving energy.

The results show that going to sleep as much as possible is critical to saving energy for the Bluetooth transceiver. However, it is hard to explicitly and immediately trigger Bluetooth sleep mode in modern smartphones. Instead, sleep mode is automatically triggered if no data has been received

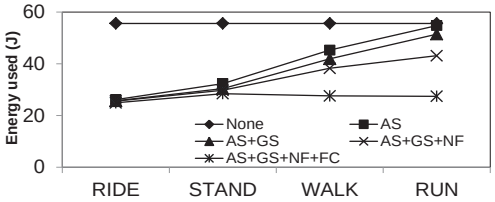


Figure 17. Energy Usage of Sensor per Mobile Situation

for a predefined time. In our case, this means that the gesture occurrence period is closely related to opportunities to enter sleep mode. Figure 16 illustrates our detailed analysis of the power consumed by the radio transceiver as the gesture occurrence period varies. It suggests that gaps between gestures of 1 sec or longer lead to more sleeping. Gaps of less than 0.4 sec are too short and provide very little savings. Fortunately, gestures in daily life are very likely to occur less frequently than every 1 sec: according to data we collected in real settings, the average gesture interval was about 13 sec, which is sufficiently apart.

5.3 Energy-efficiency per Mobility Situation

In this subsection, we demonstrate the energy-saving effectiveness of our sensor-side gesture segmentation: Accel-based Segmentor (AS), Gyro-based Segmentor (GS), Feedback Controller (FC), and mobility Noise Filter (NF). We measure the energy usages of a single sensor node for five different combinations of energy-saving methods, i.e., *AS only*, *AS+GS*, *AS+GS+NF* and *AS+GS+NF+FC*, while increasing the user's mobility. *None* is the case that the sensor device sends raw data continuously. At each setting, we measure the total energy consumption for 20 minutes.

As shown in Figure 17, we obtain significant energy saving for low-mobility situations (STAND and RIDE) by applying any combination. However, in-depth analysis reveals that AS alone dominates overall energy saving. This is because the low-mobility situations produce little noise from body movement or hand swings, which AS does not filter out. Energy consumption tends to increase with mobility, but enabling more methods cumulatively contributes to energy savings. Applying all methods consumes the least, and is hardly affected by increases in mobility. As discussed in Section 3.1, GS filters out noise caused by body movements, while NF suppresses non-gestural hand swings. Since those effects are common in high-mobility situations, we can clearly observe the energy savings of GS and NF in WALK and RUN situations.

Most importantly, FC achieves considerably more energy savings than methods without it. The essential goal of FC-based accelerometer adaptation is to maintain the FP ratio without being affected by mobility changes. Hence, FC provides near-mobility-free gesture segmentation as shown in Figure 17. We further elaborate on the benefits of FC and its on-the-fly adaptation behavior in Section 5.4.

Figure 18 shows the energy consumed by the accelerometer, gyroscope, radio transmission, and basic processing. The measurement is based on the energy consumption profile in Figure 4, i.e., 2.4mW, 26.1mW, 9.8mW, and 7.975mW, respectively. During RIDE situations (see Figure 18(a)), most energy savings is due to AS, which eliminates

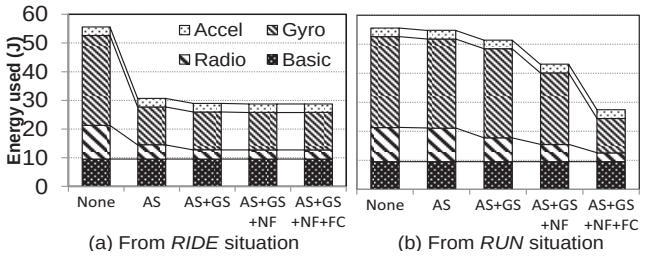


Figure 18. Per-component Energy Consumption

unnecessary operation of the gyroscope and radio transceiver. Enabling GS further halves the radio transmission cost by eliminating the transmission of invalid gesture segments. Figure 18(b) shows the RUN situation. We can see that AS has little effect and is easily overwhelmed by high mobility noises. GS is also ineffective for energy saving. Enabling NF reduces 28% of the radio transmission energy and 20% of gyroscopic operation since NF halts both the gyroscope and radio transceiver if a segment exceeds 2 secs. Finally, we can verify that FC provides well-balanced energy savings from the gyroscope (49%) and radio (52%). This is mainly because FC enhances the early filtering of AS, thus reducing the activity of the gyroscope and radio. The remaining energy consumption is necessary for gesture detection.

5.4 Run-time Threshold Adaptation

We evaluate the effectiveness of the adaptive threshold control in closed-loop collaborative segmentation. We conduct experiments to analyze how the threshold of the accel-based segmentor is adjusted under continuously changing mobility, and how both FPs and FNs are suppressed. The FPs are important because they lead to unnecessary energy consumption. FNs are important because they annoy users by rejecting normal gestures.

To this end, we construct a test workload representing continuously changing mobility situations. We use the sample data of four types of mobility situations collected in Section 5.1. In order to maintain continuity, we use the original time sequence without separating gesture samples and non-gesture ones, denoted as: $\{STAND_{1...N}\}$, $\{WALK_{1...N}\}$, $\{RUN_{1...N}\}$, and $\{RIDE_{1...N}\}$. Simply concatenating those four data sequences gives only three significant changes of mobility. To double the number of changes, we split each data sequence into halves, shuffle, and concatenate. This leaves the following test workload: $\{STAND_{1...N/2}$, $WALK_{1...N/2}$, $RUN_{1...N/2}$, $RIDE_{1...N/2}$, $RUN_{N/2+1...N}$, $WALK_{N/2+1...N}$, $RIDE_{N/2+1...N}$, $STAND_{N/2+1...N}\}$. While processing the workload, we determine whether those segments are truly gestures or FPs. Also, we count FNs, i.e., failures to detect button-annotated sections.

Figure 19 illustrates the time plot of threshold values for a user along the 75-min experiment period, as well as the fluctuation of the FP ratio. As in the figure, the FP ratio stays within the range of 8% to 20% regardless of the mobility situation. Also, thresholds are consistent with the mobility changes, i.e., high thresholds for WALK and RUN situations and low thresholds for RIDE and STAND situations. We see some latency to find the correct threshold and stabilize, less than 2 minutes in our experiment. However, most

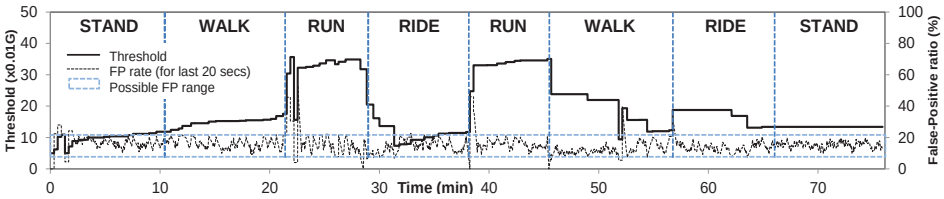


Figure 19. Time Behavior of Adaptive Threshold

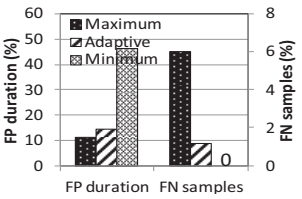


Figure 20. FP and FN

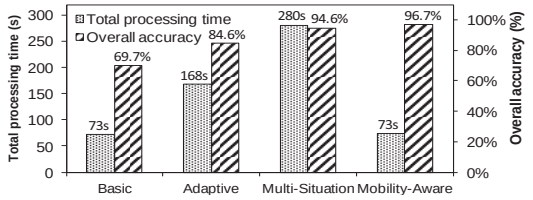


Figure 21. Accuracy and Processing Time

HMM setting	RIDE	STAND	WALK	RUN
Basic	40.3%	98.4%	87.1%	53.3%
Adaptive	81.7%	96.8%	93.0%	67.0%
Multi-Situation	95.6%	95.2%	95.7%	91.9%
Mobility-Aware	96.9%	98.4%	99.1%	92.3%

(a) Per Mobility Situation

HMM setting	Deletion error	Substitution error	Insertion error
Basic	26.99%	3.26%	6.31%
Adaptive	14.00%	1.38%	3.42%
Multi-Situation	4.21%	1.19%	2.26%
Mobility-Aware	2.76%	0.56%	2.26%

(b) Per Error Type

Table 5. Detailed Accuracy Analysis

threshold changes are just after a mobility change, thereby successfully suppressing segmentation errors resulting from an incorrect threshold.

To see the effect on segmentation accuracy of an adaptive threshold, we compare it to a naïve segmentor statically configured with the maximum and the minimum thresholds taken from the highest mobility (RUN) and the lowest (STAND), respectively. Figure 20 demonstrates that the adaptive threshold gives near-optimal results in terms of the overall FP ratio and the ratio of FN samples. Applying the maximum threshold gives the best suppression of the FP ratio (10.88%), but fails in the FN ratio by allowing 6% of FNs. Conversely, applying the minimum threshold eliminates all FNs but quadruples the FP ratio, wasting substantial energy. Note that the adaptive threshold achieves a very low (1.12%) FN ratio while keeping the FP ratio to close to 10% of the maximum threshold.

5.5 Gesture Classification Performance

In this section, we demonstrate the classification performance of E-Gesture under dynamic mobility situations by quantitatively analyzing the effects of the adaptive and multi-situation HMMs. To discuss whether our system achieves acceptable accuracy, we introduce an imaginary mobile gesture classifier, *Mobility-Aware HMM*, which is assumed to accurately infer a user's current mobility situation at no cost. Then, it classifies the gesture by using the situation-specific HMMs trained with gesture samples from the detected situation, thereby achieving the optimal accuracy. To show the baseline accuracy, we also compare the basic HMM which is trained only with the gesture samples from STAND situation. The initial training status of adaptive HMM is the same as that of basic HMM.

We test gesture classification on gesture samples while varying mobility situations, i.e., RIDE→STAND→WALK →RUN. Out of 240 gesture samples collected for each situation, we used two thirds for training, and a third for testing. Similar to other personalized gesture classifiers [15], our classification test is performed user-dependently, and results are averaged over 7 participants. We measure accuracy and processing time. The *accuracy* is the ratio of correctly classified gestures over the total number of gesture samples. The *processing time* is the total elapsed time to process all test gesture samples, and is measured on our Google Nexus One.

Figure 21 shows the classification accuracy and total processing time for four mobility situations. As expected, mobility-aware HMM shows the highest accuracy and the lowest processing time. Multi-situation HMM achieves fairly good accuracy compared to the optimal mobility-aware HMM. However, it takes four times longer to classify gestures than the mobility-aware HMM mainly due to having more HMM models. To run the multi-situation HMM even in a resource-limited mobile device, further processing optimization may be necessary. Unfortunately, adaptive HMM exhibits a low accuracy of less than 90%, which seems to be unacceptable for the purpose of mobile interaction, although it outperforms the basic HMM. This is mainly due to the lack of re-training samples, which leads to poorer fitting for each mobility situation than multi-situation HMM. The processing time of adaptive HMM is much less than that of multi-situation HMM. It is only twice as long as the basic and mobility-aware HMMs, which build a single HMM for each gesture with no adaptation costs.

Table 5(a) details classification accuracy for each mobility situation. As expected, mobility-aware HMM retains a high level of accuracy regardless of mobility situation. Most importantly, multi-situation HMMs demonstrate their robustness under multiple mobility situations even in high-mobility situations, i.e. RUN. In contrast, the accuracy of basic HMM rapidly decreases as mobility increases (STAND→WALK→RUN). In particular, basic HMM shows poor accuracy in RUN situation mainly due to elevated noise levels, i.e., only 53.3%. Interestingly, it also fails to classify accurately in RIDE situation, i.e., 40.3%. We conjecture that limited space in a RIDE situation makes the gesture samples somewhat unique, and does not match the basic HMM trained with the samples from STAND. Adaptive HMM also fails to accurately classify gestures under RIDE and RUN although it moderately improves accuracy.

Table 5(b) shows the breakdown of classification errors. We define a *deletion error* as a gesture sample that is misclassified as garbage, and a *substitution error* as a gesture sample *A* that is misclassified as gesture *B*. Note that deletion errors are present in all settings. While mobility-aware shows the lowest deletion error rate of 2.76%, most deletion errors are also eliminated in multi-situation HMM, leading

to 4.21% deletion errors. However, adaptive HMM could not reduce the deletion errors as well as the multi-situation HMM, which explains why adaptive HMM is less accurate than multi-situation HMM. In general, substitution errors occur less often than deletion errors when using a garbage model [14]. However, substitution errors are more disruptive than deletion errors and cannot be corrected by simple retrials. Note that both HMMs exhibit a fairly low substitution error rate of around 1%.

6 Discussion and Future Work

In this section, we discuss design alternatives and future extensions, as well as share our lessons in developing gesture-based UIs and applications.

Mobility detection. Instead of dynamic threshold adaption and gesture classification, one might consider an alternative architecture, i.e., detecting the current mobility situation automatically [4, 10] and using it to control the threshold and use the most appropriate HMM, respectively. However, this architecture is unlikely to be practical in our setting. First, it is difficult to accurately detect mobility situations solely based on a hand-worn sensor device. The accelerometer in the hand-worn device would not isolate body movements alone since it measures the combination of hand motion, gravity, and body movement. To accurately detect a mobility situation, we would likely require additional sensors tightly attached to the body. However, this requirement burdens users with additional devices that must be carried at all times. While mobile devices may have built-in sensors, the devices are rarely tightly attached to the body.

Comparison between gesture classifiers. The proposed HMM architectures, adaptive and multi-situation HMM, each have advantages and disadvantages relative to each other. First, adaptive HMM is advantageous because it copes well with new mobility situations, supporting a diverse and fine-grained range of situations such as slow/middle/fast walking. The processing time for adaptive HMM increases slightly relative to the basic HMM that executes MLLR upon a classification error. Unfortunately, adaptive HMM could not achieve the best accuracy under known situations, due to the scarcity of training data available at runtime that is needed to tune the parameters for each situation. Besides, error-oriented adaptation requires manual correction for misclassified gestures, a significant inconvenience that would fall on the user.

In contrast, multi-situation HMM shows the best accuracy under known mobility situations since the HMM parameters can be fit to those situations due to a large amount of offline training data. However, its processing time could be high. Since the Viterbi decoding time is proportional to the number of HMM models, multi-situation HMM takes N times longer time than the basic HMM, where N is the number of mobility situations. Also, multi-situation HMM may not respond well to new mobility situations.

We believe that our experiences provide directions for the design of future all-around mobile gesture classifiers. First, improved accuracy is required in not only diverse but unknown mobility situations. Second, burdening the user with tasks like error correction is not ideal. Third, the amount of training data needs to be as small as possible. Finally, techniques must be lightweight enough to be runnable on mobile devices. We envision that continued engineering of our current architecture will bring us closer to an ideal all-around mobile gesture classifier.

Experiences in Developing Gesture-based UI. One of our lessons in developing gesture-based UI and applications is that selecting appropriate gestures requires broad consideration of the surrounding situations in case of possible gesture-context mismatches. We discuss and share the common issues we frequently encountered.

Physical feasibility. We found that a certain gesture may not be matched to the user's physical context. For example, clapping while running can disrupt the runners' balance.

Physical availability. A user's hand may not be always free to perform a gesture. For example, clapping may be impossible while holding a heavy shopping bag.

Space allowance. Space should be considered before selecting gestures. For example, large gestures like throwing are hard to perform in a narrow space like the inside of a car.

Social acceptance. A gesture should be socially acceptable in a given situation. For example, a gesture like throwing would be inappropriate in a conference room.

Privacy concern. Users may care about privacy; they may mind exposing what they are doing to others. Needlessly large gestures in public places to check an SMS will let others know that the user is doing so.

Intuitiveness. A gesture should be intuitively coupled to its intended function. Users maybe be confused if the gesture of drawing a circle is used to transmit a message.

Inter-gesture distinction. The gesture vocabularies should be carefully chosen to be sufficiently distinct from one another. We have experienced that a certain combination of gestures like {left knob, left slide} caused frequent confusion, yielding insufficient precision of 88%.

7 Conclusion

In this paper, we have presented E-Gesture, an energy-efficient, accurate gesture recognition framework for dynamic mobility situations. Spanning a wearable sensor device and an off-the-shelf mobile device, E-Gesture recognizes the gesture through closed-loop collaborative segmentation and multi-situation HMM classification. Evaluation with seven users in four mobility situations shows that it achieves stable classification accuracy, i.e., 94.6%, while reducing the energy consumption of a sensor and a mobile device by 58.7% and 64.3%, respectively.

Acknowledgments

Special thanks to our shepherd, Professor Landon Cox, and all the anonymous reviewers for their great reviews and suggestions to improve the quality of this paper. This work was supported by the National Research Foundation of Korea under grants No. 2011-0018120 and No. 2011-0020519.

8 References

[1] HMM toolkit (HTK). http://htk.eng.cam.ac.uk/.

[2] R. Amstutz,O. Amft, B. French, A. Smailagic, D. Siewiorek, and G. Troster. Performance analysis of an hmm-based ges-

ture recognition using a wristwatch device. In *Proceedings of the International Conference on Computational Science and Engineering*, pages 303–309. IEEE, 2009.

[3] W. Bang, W. Chang, K. Kang, E. Choi, A. Potanin, and D. Kim. Self-contained spatial input device for wearable computers. In Proceedings of *the IEEE International Symposium on Wearable Computers*, page 26. IEEE, 2003.

[4] L. Bao and S. Intille. Activity recognition from user-annotated acceleration data. In *Pervasive Computing*, volume 3001 of *LNCS*, pages 1–17. Springer, 2004.

[5] A. Benbasat and J. Paradiso. An inertial measurement framework for gesture recognition and applications. In *Gesture and Sign Language in Human-Computer Interaction*, volume 2298 of LNCS, pages 77–90. Springer, 2002.

[6] R. Ganti, P. Jayachandran, T. Abdelzaher, and J. Stankovic. Satire: a software architecture for smart attire. In *Proceedings of the 4th international conference on Mobile systems, applications and services*, MobiSys '06, pages 110–123. ACM, 2006.

[7] A. Haro, K. Mori, T. Capin, and S. Wilkinson. Mobile camera-based user interaction. In *Proceedings of Computer vision in human-computer interaction Workshop*, page 79, Springer, 2005.

[8] H. Junker, O. Amft, P. Lukowicz, and G. Troster. Gesture spotting with body-worn inertial sensors to detect user activities. *Pattern Recognition*, 41(6):2010–2024, 2008.

[9] S. Kang, S.S. Iyengar, Y. Lee, J. Lee, C. Min, Y. Ju, T. Park, Y. Rhee, and J. Song. Mobicon: Mobile context monitoring platform for sensor-rich dynamic environments. To appear in *Commun. ACM*, 2011.

[10] S. Kang, J. Lee, H. Jang, H. Lee, Y. Lee, S. Park, T. Park, and J. Song. Seemon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, MobiSys '08, pages 267–280. ACM, 2008.

[11] S. Kang, Y. Lee, C. Min, Y. Ju, T. Park, J. Lee, Y. Rhee, and J. Song. Orchestrator: An active resource orchestration framework for mobile context monitoring in sensor-rich mobile environments. In *Proceedings of the 8th Annual IEEE International Conference on Pervasive Computing and Communications*, pages 135–144.

[12] J. Kela, P. Korpipaa, J. Mantyjarvi, S. Kallio, G. Savino, L. Jozzo, and S. Marca. Accelerometer-based gesture control for a design environment. *Personal and Ubiquitous Computing*, 10(5):285–299, 2006.

[13] J. Kim, J. He, K. Lyons, and T. Starner. The gesture watch: A wireless contact-free gesture based wrist interface. In *Proceedings of the IEEE International Symposium on Wearable Computers*, pages 1–8. IEEE, 2007.

[14] H. Lee and J. Kim. An hmm-based threshold model approach for gesture recognition. 21(10):961–973, 1999.

[15] J. Liu, Z. Wang, L. Zhong, J. Wickramasuriya, and V. Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. In Proceedings of the Annual IEEE International Conference on Pervasive Computing and Communications, pages 1–9. IEEE, 2009.

[16] K. Lorincz, B. Chen, G. Challen, A. Chowdhury, S. Patel, P. Bonato, and M. Welsh. Mercury: a wearable sensor network platform for high-fidelity motion analysis. In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, pages 183–196. ACM, 2009.

[17] K. Lyons, H. Brashear, T. Westeyn, J. Kim, and T. Starner. Gart: The gesture and activity recognition toolkit. In Proceedings of the international conference on Human-computer interaction, pages 718–727. Springer, 2007.

[18] Nintendo. Nintendo wii. http://www.nintendo.com/wii/.

[19] T. Pering, Y. Anokwa, and R. Want. Gesture connect: facilitating tangible interaction with a flick of the wrist. In Proceedings of the international conference on Tangible and embedded interaction, pages 259–262. ACM, 2007.

[20] T. Pering, P. Zhang, R. Chaudhri, Y. Anokwa, and R. Want. The psi board: Realizing a phone-centric body sensor network. In *Proceedings of the 4th International Workshop on Wearable and Implantable Body Sensor Networks*, BSN '07, pages 53–58. Springer, 2007.

[21] G. Raffa, J. Lee, L. Nachman, and J. Song. Don't slow me down: Bringing energy efficiency to continuous gesture recognition. In Proceedings of International Symposium on Wearable Computers, pages 1–8. IEEE, 2010.

[22] T. Schlomer, B. Poppinga, N. Henze, and S. Boll. Gesture recognition with a wii controller. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, TEI '08, pages 11–14. ACM, 2008.

[23] T. Stiefmeier, D. Roggen, and G. Troster. Gestures are strings: efficient online gesture spotting and classification using string matching. In *Proceedings of the ICST 2nd international conference on Body area networks*, pages 1–8. ICST, 2007.

[24] Y. Wang, J. Lin, M. Annavaram, Q. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, MobiSys '09, pages 179–192. ACM, 2009.

[25] A. Wilson and S. Shafer. Xwand: Ui for intelligent spaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '03, pages 545–552, New York, NY, USA, 2003. ACM.

[26] Y. Wu and T. Huang. Vision-based gesture recognition: A review. In *Proceedings of the International Gesture Workshop*, GW '99, page 103. Springer, March 1999.