Energy-Efficient Mobile Gesture Recognition with Computation Offloading

Noura Farra[1], Giuseppe Raffa[2], Lama Nachman[2] and Hazem Hajj[1]

[1]Department of Electrical and Computer Engineering
American University of Beirut
Beirut, Lebanon
{naf08, hazem.hajj}@aub.edu.lb

[2]Interaction and Experience Research
Intel Labs, Intel Corporation
Santa Clara, CA (USA)
{giuseppe.raffa, lama.nachman}@intel.com

***Abstract*— Gesture recognition is a novel and compelling user input modality which allows users to interact quickly and naturally with their devices with less demand on their visual attention. Continuous gesture recognition places stringent demands on device power consumption, battery life and processing capability. In this work, we show that we can reduce the energy consumed during continuous gesture recognition on a mobile device with the delegation of the pre-processing stages, which filter out non-gesture segments, to a low power node that is separate from the main CPU. The main CPU can thus be kept in stop mode until a potential gesture is detected by the low power node, invoking the main processor to perform the computation-intensive gesture classification to detect which exact gesture has been performed by the user. We present details of the processing performance and power consumed at each step of the processing pipeline, showing the extent of power savings achieved. Experiments were conducted for detailed evaluation of the power consumption of the optimized gesture pipeline.**

Keywords-continuous gesture recognition, energy-efficiency, movement detection, template matching, 3D accelerometer and 3D gyroscope, HMM

I. INTRODUCTION

Gesture recognition on mobile and hand-held devices is a compelling interaction modality that allows users to interact with their devices on-the-go without having to pay visual attention to or be limited by a traditional interface such as keyboard, textual interface or touch screen. Detected gestures can trigger functions from any gesture-enabled application that is running such as music player, games, calendar, voice mail, phone calls, etc. Gesture recognition is part of a larger computing paradigm aimed at making devices aware of their users' context and intentions hence enabling more natural human-machine interaction.

Like other context-sensing applications, gesture recognition requires continuous capture of data from sensors – which places stringent demands on the device power consumption, battery life and processing capability. Continuous capture and filtering of non-gestural data from the sensors force the main CPU to stay active consuming power while waiting for a true gesture to be performed. While most existing work relies on having the user explicitly indicate the start and stop of gestures to improve accuracy and reduce computational overhead, we found that such a solution is cumbersome from a usability perspective. In this work, we delegate the preprocessing tasks to a low-power processing node that enables the main CPU to go into a stop mode until a potential gesture has been detected.

In previous work [1], we developed an efficient pipeline for continuous gesture recognition based on data from 3D accelerometer and gyroscope sensors and we proved that adding a gyroscope results in accuracy improvement over an accel only approach. The system was designed to automatically detect potential gestures from continuous input data streams. The front-end consisted of low computation pre-processing stages which classify gestural segment data from non-gestural movements based on accelerometer features. The back-end consisted of the computation-intensive Hidden Markov Model (HMM) and rejection algorithms responsible for classifying the specific gesture from a set of trained gestural data, leveraging both gyroscope and accelerometer. While the front-end needs to run continuously, the back-end only needs to run when a potential gesture is detected.

While such partitioning is advantageous from a computation and power perspective, since compute intensive stages are invoked sparingly, more power savings can be realized by offloading these low compute stages to low power micro-controllers as they are more suited for such simple tasks. Our contributions can be summarized as follows: (1) we developed a low power sensing subsystem that consists of sensors connected to a low power micro-controller which interfaces to the application processor of a mobile device, (2) We effectively partitioned the processing pipeline for continuous gesture recognition across the main CPU and the low power processor and (3) We provide details of the processing time and power consumption at every stage.

In section II, we discuss related work in gesture recognition on mobile devices. In section III, we describe the gesture recognition pipeline and our approach to energy-efficient gesture recognition by offloading the movement detection and early template matching stages to the low power processing

node. In section IV we present details of the performance and power analysis at each stage of the pipeline. We conclude our findings in section V and discuss future work.

II. Related work

Several works in gesture recognition have used camera and vision-based recognition [2, 3] or camera-enabled mobile phones [4]. Such systems are not yet practical in real mobile environments due to limited wearability and high demands on computation and energy. On the other hand, we focus on accelerometer and gyroscope based gesture recognition which is more practically usable in a real on-the-go environment. Such systems that have been successfully implemented include the Wii Controller [5], VTT Soapbox [6,7], GeorgiaTech Watch [8], KAIST Ring [9], and ETH Wearable Computer [10]. In [5] and [8] the system relies on the user explicitly indicating the start and stop of gestures which is inconvenient from a usability perspective. VTT Soapbox [6,7] utilizes discrete HMM and vector quantization, impacting accuracy. In [9] only four gestures are considered. In [10] gesture spotting is discussed but the authors did not consider energy efficiency or use in mobile settings. A few systems which have been used to automatically detect gestures from continuous sensor streams are found in [9,10,11,12]. However, these have shown to have either higher computation costs such as HMM-based methods in [12] and analysis-based methods in [10], or lower filtering rates in threshold-based methods such as in [9,11]. In [13], authors developed watch-based recognition system based on discrete HMM. However, the vocabulary is limited to three gestures with low accuracy, i.e. 93.5%. Furthermore, none of these systems explored offloading of the pre-processing stages or examined energy consumption in detail.

From an algorithm perspective, the most used gesture recognition algorithms are HMM [14], SVM [16], DTW [15] and Bayesian Networks [17]. The HMM algorithm has been so far reported to have the best accuracy (up to 98.5%) [14], but it is the most computationally expensive.

In our work in [1], we adopted a computation-intensive HMM-based back-end achieving an average gesture recognition accuracy of 98% and a threshold-based pre-processing segmentation method with a 4% false positive and 0.1% false negative filtering rates. The system used 3D accelerometer and gyroscope based gesture recognition and was implemented on a sensor-enabled wrist watch which communicated with a mobile device via Bluetooth. The reader is referred to [1] for detailed approach and experimental results. In this work, we implement the entire system on an experimental mobile device and offload the pre-processing stages to a low-power sensing node and evaluate the corresponding energy savings.

III. Approach to energy-efficient gesture recognition

In this section we describe our approach to energy-efficient gesture recognition by briefly introducing the gesture recognition pipeline and describing how we offloaded the front-end pre-processing to the low power microcontroller.

The pre-processing stages filter out non-movement and non-gesture data from potential true gestures to be classified by the HMM algorithm. By partitioning the processing and offloading the front end, we realize the following benefits :

(1) Reduced data load on the HMM algorithm because we drop most non-gestural data before the HMM stage

(2) Reduced power consumption during continuous recognition by allowing the main CPU to go to stop mode instead of having to stay in active mode to capture data and pre-process each sample

We describe below the architecture of the system, followed by a brief description of the end-to-end gesture pipeline as well as the pipeline partitioning between the two processors.

A. System Architecture

We used an internal prototype of a mobile device equipped with an Intel Atom CPU which operates at a clock frequency up to 1.9 GHz ("Moorestown" platform). The mobile device contains an embedded low-power microcontroller that interfaces with the 3D accelerometer and 3D gyroscope (among other sensors) and communicates with the main Intel Atom CPU via an SPI port and GPIOs. The low power node (LPN) microcontroller is a Cortex M3 processor that can operate at a variable frequency up to 72 MHz. In the present work, the LPN has been run at 8MHz, sufficient to implement the front end algorithms. A diagram of the system is shown in Figure 1.

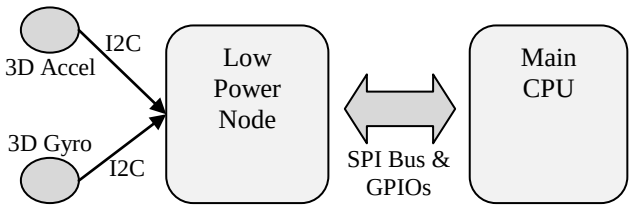

Figure 1: System Architecture of low-power gesture recognition system

B. Gesture pipeline

The gesture pipeline we developed consists of the following stages: (1) Sensors capture, (2) low pass filtering, (3) Movement Detection, (4) Early Template Matching, (5) Normalization, (6) Feature Extraction, (7) HMM and Garbage Model, and (8) Late Template Matching.

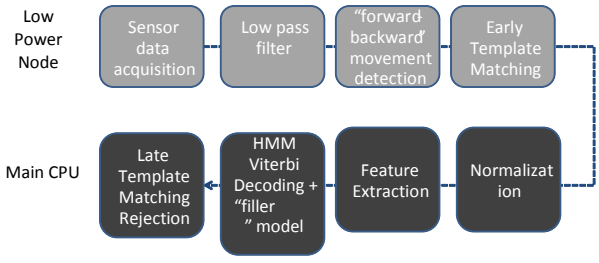

Figure 2: Block diagram of partitioned gesture recognition processing system

Sponsored by the Intel-Middle East Energy Efficiency Research (MER) Program

Stages 1 – 4 constitute the data capture and front end pre-processing and are offloaded to the LPN. Stages 5-8 constitute the computation-intensive back end processing and run on the Intel Atom CPU. A block diagram of the partitioned processing system is shown in Figure 2.

In stage (1), data is captured from the 3D accelerometer sensors at a sampling rate of 100 Hz. To reduce power consumption, data capture from the power hungry 3D gyroscope is only triggered when movement from the user is detected based on accelerometer features. Low-pass filtering using an Exponential Moving Average filter is subsequently applied to the signal to eliminate high frequency noise and sensor noise.

In the Movement Detection (MD) stage, accelerometer-based segmentation is used to detect a movement from the user carrying the device. This stage runs after every capture of an accelerometer sample, buffering samples when movements are being detected. In case a complete movement segment is detected, the Early Template Matching (ETM) stage is triggered to determine if the movement is a gesture or not.

If a movement is classified as a possible gesture by the early template matching stage, the gesture 6-dimensional data buffer (3D accel and 3D gyro raw data) is forwarded to the back-end pipeline for specific gesture recognition by the HMM and garbage model (HMM + G), followed by a late template matching stage (LTM) to further reduce the false positives.

Our implementation yielded a 96% discard rate and 4% false positive rate with the early template matching stage. An average recognition accuracy of 88% was obtained after the HMM + G model and a final gesture recognition accuracy of 98% was obtained after the late template matching model with 4% false positives and 0.1% false negatives.

C. Offloading Moving Detection

The movement detection stage is based on calculation of the 'Hand Force' from each accelerometer sample:

$$HF = Abs\left(\sqrt{(a_x^2 + a_y^2 + a_z^2)} - g\right) \qquad (1)$$

where (a_x, a_y, a_z) is a 3D accelerometer sample and g is the gravitational constant.

When the hand force (filtered by a low pass filter) meets a certain strong threshold HF_{tS}, the movement detection enters the 'during gesture' state, and consequently triggers capture of data from the gyroscope. The 'during gesture' state ends when a certain forward threshold HF_{tF} is met, and the resulting data is sent to the early template matching stage. The implementation of the movement detection is based on Forward Backward Movement Detection (FBMD) in [1];

The Movement Detection stage should run every 10 ms after capture of each sample from the accelerometer. If this stage were to be run on the main CPU, the processor would be forced to transition between active (C_0) and sleep (C_6) states at every 10 ms for each sensor (accel and gyro), consuming unnecessary active power to sample the sensor data using the slow I2C bus as well as to process each sample. On the other hand, offloading the MD stage to the LPN would reduce this unnecessary overhead. Moreover, it would enable the main CPU to transition to a deeper standby state (S3) if no other applications require the CPU to be active.

We implemented the Movement Detection stage on the LPN, where data was being captured and processed from the accelerometer at 100 Hz. The LPN can be duty cycled between stop and active mode whereby it can wake up to capture an accelerometer sample and process it, and sleep in between.

D. Offloading Early Template Matching

In the Early Template Matching (ETM) stage, a preliminary binary classification of gesture/non-gesture is implemented based on comparing the movement data (potential gesture data) with the gestural training data, optimized for minimizing false negatives. We chose to focus on eliminating false negatives as much as possible at the expense of false positives because the former will result in missed recognitions while the latter can be caught and filtered by the backend be it at a higher energy cost. A min-max bounding box is calculated for each feature using the training data from all gestures. Then, in the template matching, for each potential gesture, the features are compared to the bounding box. The features considered in our implementation were the movement duration T and the maximum hand force *MaxForce* of the movement. From the training data of all gestures we can infer the two bounding boxes for duration $[T_{MIN}, T_{MAX}]$ and MaxForce $[F_{Min}, F_{Max}]$, as following:

$$MaxForce_{G_j} = Max_{t_1}^{t_N}\left(HF_{t_i}\right) \qquad \forall\, j \in [1, M] \qquad (2)$$

$$F_{MIN} = min_{G_1}^{G_M}(MaxForce_{G_j}) \qquad (3)$$

$$F_{MAX} = max_{G_1}^{G_M}(MaxForce_{G_j}) \qquad (4)$$

Similarly, we implemented the early template matching ETM stage on the LPN and measured the processing time for different gesture sizes at different clock frequencies. The ETM stage is triggered after a potential gesture is identified by the MD stage and its output is a binary classification of 'gesture/non-gesture', whereby an event is triggered signaling the initiation of the back-end pipeline on the main processor to classify the specific gesture.

IV. Evaluating Energy Consumption

In this section we evaluate energy consumption by presenting the duty cycle, power consumption, and processing time at each of the processing stages and we show the advantage of task offloading versus running the gesture recognition pipeline on the main CPU.

Table 1 shows the power consumption of the Cortex M3 during active and stop.

TABLE I. AVERAGE POWER CONSUMPTION ON CORTEX M3

Power Consumption (mW)	Active	Stop
Cortex M3	25	0.13

A. *Movement Detection Performance and Power*

1) Duty Cycle and Processing Time

We measured the duty cycle of the LPN between active and stop while capturing data from the accelerometer at 100 Hz and subsequently performing the movement detection stage.

The duty cycle τ is defined as the percentage of time the processor is actively consuming power, whereby:

$$P_{avg} = \tau * P_{active} + (1- \tau)*P_{stop} \quad (5)$$

Table II shows the duty cycle and processing time for data capture from the sensors and movement detection processing when the sensors are sampled at 100 Hz (total period of 10 ms). Measurements were obtained for both accelerometer and gyroscope to represent both the case of no movement (only accel is sampled) and "during movement", when accel and gyro are both sampled.

It is worth noting that data capture from the gyroscope only occurs in the 'during gesture' phase; hence the measurements representing both accelerometer and gyroscope capture are a worst case scenario. The duty cycle and corresponding power consumption will thus depend on the fraction of time that the gyroscope is being captured which in turn depends on the fraction of time that a movement is detected from the user.

TABLE II. MOVEMENT DETECTION PERFORMANCE ON LOW-POWER NODE

Clock Frequency, Sensor Capture	MD Processing (ms)	Duty Cycle MD (%)	Sensor Capture (ms)	Duty Cycle Capture (%)
8 MHz, accel	0.368	3.68 %	1.8	18%
8 MHz, accel + gyro	0.368	3.68%	3.1	31%
72 MHz, accel	0.0552	0.55%	0.96	9.6%
72 MHz, accel + gyro	0.0552	0.55%	1.8	18%

2) Power Consumption

We measured the power consumed during accelerometer and gyroscope capture and Movement Detection stages on the LPN at 8 MHz, as shown in Table III.

The average power consumed is determined according to equation (5).

TABLE III. AVERAGE POWER CONSUMPTION ON LPN (8MHZ)

	Active Power (mW)	Capture Duty Cycle	Average Power Consumption (mW)
Accel Capture+MD	26.37	21.68%	5.8
Accel + Gyro Capture+MD	45.66	34.68%	15.9

A. *Early Template Matching Performance*

Early Template Matching stage (ETM) running at 8 MHz on the LPN, shows a processing time linear with gesture size, with average processing time in the range of 57 msec per typical gesture.

In terms of duty cycle, since ETM is performed sporadically, only once per gesture, the ETM computation affects only marginally the LPN duty cycle. For example, for a typical gesture of 1 second, the incurred computation is 35 msec, or 1.75% duty cycle in the worst case.

B. *Performance and Power on main CPU*

So far we have shown the performance in terms of average power consumed and processing times of MD and ETM running on the low power node. We now see how the processing pipeline would behave when exclusively run on the main CPU.

Table V shows the processing time of running the entire pipeline on the main CPU with no offloading. Two gestures 'flick and circle' are considered.

TABLE V. GESTURE PIPELINE PERFORMANCE ON MAIN CPU

Processing Time (ms)	'flick' -58 samples	'circle'-109 samples
MD	0.01*58→0.58	0.01*109 →1.09
Early TM	0.26	0.32
Normalization	3.16	4.49
Feature Extraction	0.99	0.77
HMM Prep (copy/format data)	0.28	0.28
HMM	16.72	14.74

Late TM	0.25	0.24
Total Duration	**21.68**	**20.86**

As can be seen from Table V, the early stages consume about 1.4% of the total pipeline running time on the main CPU (per gesture) but they will run most of time in a typical scenario where movement represents the majority of the data and gestures are assumed to represent the minority of the data. In [1] we showed an average false positive rate of MD of ~28.8% but only 4% for the ETM, hence on average all the gesture data + 4% of the movement data will flow to the main CPU. The HMM algorithm is the most time consuming stage but it will run only once per gesture performed and not continuously. Hence, offloading the initial stages we will considerably limit the time the main CPU will be active.

C. Energy Advantage of Offloading the Early Stages

In this section we compare the power consumption in the two cases of the initial stages of the pipeline implemented in the main CPU compared with the optimized version on the LPN.

Consider the case where the full pipeline runs on the main CPU and no offloading occurs. The CPU will be consuming "active" power when it is waiting for a sample and "sleep" power when it captures and processes a sample. We are in the process of obtaining those numbers.

Energy consumed per gesture can be decomposed in three main components: (1) Capture and MD time, once per sample, (2) ETM time, once per gesture, (3) energy consumed during sleep/stop mode[1]:

$$E_{gesture} = (T_{wakeup}+T_{capture}+T_{MD}+T_{go\_to\_sleep})*P_{active}*N_{samples}+ T_{ETM}*P_{active} +P_{sleep/stop}* T_{sleep/stop} \quad (6)$$

Considering the "circle" gesture (1.09sec duration), considering the pipeline running on LPN, the energy utilized per gesture would be:

(0.0054+3.1+0.368+0.0054)*25*109 +
35*25+
(10-(0.0054+3.1+0.368+0.0054))* 0.13 *109 = 10.44 mJ

Similarly, when no gesture or movement is occurring, the implementation on LPN only consume 6.05 mJ. When gesture spotting is implemented on LPN, we can take full advantage of fast transitions between active and stop modes of LPN instead of the much longer entry/exit latency times of application processors, typically usec in LPN vs. msec on Main CPU. Hence, LPN can be put in stop mode between sensors acquisitions (with power consumption in the order of uW) and Main CPU can be kept in more energy efficient power states (with power consumption in the order of uW or few mW) instead of the more power consuming but faster sleep states, necessary when the main CPU actively polls data from sensors.

In both cases of gestures performed and no gestures, we can expect a considerable power saving.[2]

Such a scenario would occur if the running application is not CPU-intensive. In the case of a music player scenario, for example, the main CPU can be kept completely in energy efficient Stop mode until a gesture occurs (as audio playback can occur through a dedicated audio path). Maximum energy savings would be incurred in this case. In general, the percentage of CPU usage while running the gesture-enabled application will impact the extent of energy savings achieved.

D. Dependence on Gesture Frequency

The extent of energy savings achieved is also highly dependent on the frequency of time that the user performs a gesture. In the first extreme, maximum energy savings are incurred when the user does not gesture at all: in this case the mobile device remains in sleep mode and no computation-intensive HMM processing occurs, and the overall power of the system is determined only by the power consumed by the low-power microcontroller. The energy efficiency will decrease as the frequency of gestures increases; the main CPU will spend more time in active mode and less time in stop mode. However, it is expected that power savings will be incurred in most cases as we assume that on average users will not be performing gestures continuously, but it will be a sparse interaction modality in typical on the go applications such as Mp3, voice mail, etc... More detailed experiments are needed to verify this assumption.

E. Smart Sensors

New sensor modules are currently available that allow a certain amount of data buffering and basic processing / threshold detection. While these capabilities are great for reducing power consumption as the main CPU can be interrupted less often, it is important to note that an LPN will still provide extra power savings in such an application over directly connecting the smart sensor to the main CPU. To understand why this is the case, consider the following two possibilities. One possibility is to push some basic form of movement detection to the sensor itself and eliminate the LPN, however, in this case the false positives will increase dramatically if we only perform movement detection without template matching in low power. A better solution from a power perspective would be to push this basic movement detection to the sensor but still run a more accurate gesture spotter on the LPN. This allows the LPN to stay in stop mode

[1] All the power numbers and transition times are obtained from official datasheet and specifications

[2] We are in the process of measuring power numbers and latency times for a typical smartphone application processor. They will be submitted with the final camera ready version of the paper

longer and trigger the switch to active mode only when some movement is detected.

Another possibility would be to buffer 100s of msec worth of data in the sensor and interrupt the main CPU less often again without having an LPN. This reduces the number of interrupts that the main CPU will handle hence improving power. Again, this is not ideal for two reasons. The first is that the gyro will need to be captured all the time since we can't trigger the gyro acquisition in real-time. The power consumption of the gyro sensor in this case is higher than the power consumption of the LPN. The second issue is that to really see considerable improvement in transition power consumption, we need to increase the buffering time considerably which will cause a latency problem for the recognition.

V. Conclusions and Future Work

In this work, we partitioned the processing pipeline for continuous mobile gesture recognition across the main CPU and a low-power processing node and showed why such a partitioning results in significant energy savings without compromising system accuracy and performance. We provided details of the processing performance and power consumption at each stage and based on qualitative analysis, showed that the extent of power savings depends on the main CPU usage and on the percentage of time that the user performs a gesture. To the best of our knowledge no other work has examined energy analysis for mobile gesture recognition by computation offloading. In future work, we plan to enact specific gesture recognition usage scenarios with different running applications whereby exact power savings can be determined in each case.

VI. Acknowledgment

This work was partially funded by the Intel-Middle East Energy Efficiency (MER) program.

VII. References

[1] G. Raffa, J.Lee. L. Nachman, and J. Song, "Don't Slow Me Down: Bringing Energy Efficiency to Continuous Gesture Recognition"

[2] Y. Wu, T. S. Huang, "Vision-Based Gesture Recognition: A Review", In *Proc. of Gesture-Based* Communication in Human-Computer Interaction, 1999

[3] H. Brashear et al, "Using Multiple Sensors for Mobile Sign Language Recognition", In *Proc. of ISWC* 2003

[4] Jingtao Wang, Shumin Zhai, and John Canny. 2006. Camera phone based motion sensing: interaction techniques, applications and performance study. In Proceedings of the 19th annual ACM symposium on User interface software and technology (UIST '06). ACM, New York, NY, USA, 101-110. DOI=10.1145/1166253.1166270 http://doi.acm.org/10.1145/1166253.1166270

[5] T. Schlomer et al, "Gesture Recognition with a Wii Controller", In Proc. of TEI, 2008

[6] J. Kela et al, "Accelerometer-based gesture control for a design environment", Personal and Ubiquitous Computing Journal, 2006

[7] J. Mantyjarvi, J. Kela, P. Korpipaa and S. Kallio, "Enabling fast and effortless customization in accelerometer based gesture interaction", In Proc. of MUM, 2004

[8] K. Lyons et al, "GART: The Gesture and Activity Recognition Toolkit, In Proc. of HCI, 2007

[9] J. Yoo,W. Hwang, S. Baek, and K. Park, "Intuitive Interface device for Wearable Computers", Next Generation PC, 2005

[10] H. Junker, O. Amft, P. Lukowicz, and G. Troster "Gesture spotting with body-worn inertial sensors to detect user activities", Pattern Recognition Journal, 2007

[11] F. Hofmann et al, "Velocity profile based recognition of dynamic gestures with discrete Hidden Markov Models" In Proc. of Gesture and Sign Language in HCI, 1998

[12] H. Lee and J. Kim, "An HMM-Based Threshold Model Approach for Gesture Recognition", IEEE Transactions on Pattern Analysis and Machine Intelligence, 1999

[13] R. Amstutz, O. Amft, B. French, A.Smailagic, D.P. Siewiorek, G.Tröster: Performance Analysis of an HMM-Based Gesture Recognition Using a Wristwatch Device. CSE (2) 2009: 303-309

[14] T. Pylvannainen, "Accelerometer Based Gesture Recognition Using Continuous HMMs", In Proc. of IbPRIA 2005]

[15] J. Lin et al, "uWave: Accelerometer-based Personalized Gesture Recognition and Its Applications", PerCom, 2009

[16] Jiahui Wu, Gang Pan, Daqing Zhang, Guande Qi, and Shijian Li. 2009. Gesture Recognition with a 3-D Accelerometer. In Proceedings of the 6th International Conference on Ubiquitous Intelligence and Computing (UIC '09), Daqing Zhang, Marius Portmann, Ah-Hwee Tan, and Jadwiga Indulska (Eds.). Springer-Verlag, Berlin, Heidelberg, 25-38

[17] S. Cho et al, "Two-stage Recognition of Raw Acceleration Signals for 3-D Gesture-Understanding Cell Phones", In Proc. of IWFHR, 2006