A 2-D Motion Detection Model for Low-Cost Embedded Reconfigurable I/O Devices

Apostolos Dollas*, *Senior Member, IEEE*, Stamatios Sotiropoulos, and Kyprianos Papademetriou

***Abstract*—A low-cost reconfigurable embedded apparatus for two-dimensional (2-D) motion detection has been developed. This paper briefly outlines the embedded reconfigurable system architecture, and presents in-depth the 2-D motion detection model, which is directly mapped to reconfigurable hardware. Emphasis is placed on the hardware ability to adapt to individual needs of kinetically challenged persons by altering detection thresholds and delays, thus resulting into an efficient low-cost reconfigurable hardware implementation of the model. This paper also presents how the model detects complex motions through a vocabulary of simple motions, and how the system is trained to individual users' needs. Experimental results and integrated applications of the model for text processing are also presented.**

***Index Terms*—Input/output (I/O) device, kinetically challenged, reconfigurable embedded system, 2-D motion detection.**

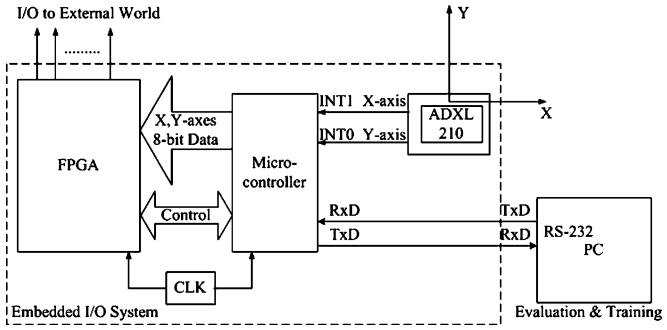

Fig. 1. Architecture of the embedded system. The system comprises of an FPGA, a microcontroller and a digital two-axis accelerometer. The PC is used for evaluation and training only.

I. Introduction

THE PROBLEM of motion detection and recognition has been considered from a number of perspectives, ranging from input/output (I/O) for virtual reality environments to gesture recognition systems. Similarly, the problem of I/O devices for kinetically challenged persons has been addressed from a mechanical design perspective to a brain activity detection perspective. In this paper we present a *low-cost, embedded I/O* device for kinetically challenged persons. The ultimate purpose of this paper is to have *shrink-wrapped hardware* which can be customized and retrained to individual user needs without a recompilation of the design.

Various assistive devices have been developed for persons with motor disabilities [1], [2]. Devices related to the control of a wheelchair [3], projects using virtual reality technology [4] and camera-based systems [5], [6] have been implemented for rehabilitation purposes. Several projects use reconfigurable logic devices, such as field programmable gate arrays (FPGAs) [7], which undertake the control of the system [8]–[10]. It should be noted that essentially all of the above projects use at least one of the following: 1) large or multiple FPGAs; 2) PC-class fixed computer resources; or 3) expensive equipment to make all necessary data acquisition and calculations. In our system, such approaches would not meet cost, size, and power consumption limits for an embedded device, i.e., a portable device that can exist and operate in a standalone mode.

Section II has an outline of the system architecture. Section III presents the model for motion detection. Section IV introduces the need of system adaptation to the user and Section V describes the unsupervised training method. Section VI presents the experimental results. Finally, Section VII presents conclusions from this work as well as its future directions.

II. System Architecture

Various alternatives (goniometric, optoelectric, and accelerometric) regarding hand motion measurement, have been proposed [11]. Our system is based on accelerometers to sense hand motion in 2-D space. The general characteristics of the system are: 1) low-cost solid-state (nonmechanical) sensors which are reliable and suitable for motion sensing; 2) real-time sampling of the data; and 3) motion detection for a large number of motions (vocabulary), tunable to the needs of different persons.

Regarding the computational subsystem, we showed in a previous publication [12] that a model of independently operating finite state machines (FSM) offers a good design tradeoff versus the usage of microcontrollers alone for free space motion detection. Furthermore, we showed that the sampling of real-time data is best performed by microcontrollers, leading to a hardware organization with fixed, as well as reconfigurable resources. Having the aforementioned observations in mind, we concluded to the architecture shown in Fig. 1.

A digital two-axis accelerometer (Analog Devices ADXL210) is attached to the user's hand to acquire data during hand motions. An 8-bit microcontroller (ATMEL AVR 90S8515) has been used for the sampling of sensor data in real

Manuscript received July 4, 2003; revised November 27, 2004. This work was supported in part by the Greek Secretariat of Research and Technology (GSRT) and the British Council, under the Britain Greece Joint Research Program, and in part by the EPET II European Union program under the Second Framework of Support to Greece. *Asterisk indicates corresponding author.*

*A. Dollas is with the Microprocessor and Hardware Laboratory, Electronic and Computer Engineering Department, Technical University of Crete, 73100 Chania, Greece (e-mail: dollas@mhl.tuc.gr).

S. Sotiropoulos and K. Papademetriou are with the Microprocessor and Hardware Laboratory, Electronic and Computer Engineering Department, Technical University of Crete, 73100 Chania, Greece (e-mail: sotirop@mhl.tuc.gr; kpapadim@mhl.tuc.gr).

Digital Object Identifier 10.1109/TBME.2005.851485

time. Then, the data are sent to an FPGA (Xilinx SpartanII XC2S100) via an 8-bit bus and to a PC via the serial port. The purpose of the embedded FPGA is to distinguish types of motions from a programmable vocabulary. The ability to connect with the PC was employed as a rapid prototyping tool for algorithm evaluation, and as a user interface for system training (to adapt to individual needs), but not as a necessary component during field deployment. The total system cost is less than $70, an arbitrary limit, which nonetheless precluded certain types of solutions to the motion detection problem.

The accelerometer is calibrated when it is powered up. This way, the system can be adjusted to different orientations and also to different accelerometers (there are minor differences due to manufacturing variations), when the original sensor has to be replaced. Moreover, the system can operate in different temperature conditions.

III. Motion Detection Model

The computational model of the system is that of parallel FSM, each of which comprises of stages for detection of values/ranges of X-Y data, followed by stages to wait for a predefined period of time (including zero time). This way each motion is represented in terms of thresholds, which needs to be exceeded for the state to be active, followed by periods of "not examining the input," which are useful to avoid local minima (from irregular motion or noise).

During preliminary clinical evaluation of the system, many experiments have been performed, and thirteen motions have been successfully sampled and processed [12]. However, it turned out that a large number of complex motions was undesirable to the user, regardless of system capability. For example, circular motions are more complicated than basic motions (such as "left," or "forward") and they require more effort in order to be performed. A simple vocabulary of motions that can be used in succession can lead into more options for the user. Therefore, the model to which we concluded, is based on the concept that sequences of the four simple motions (forward, back, left, right) are used in order to produce a complex "vocabulary" [13]. This approach allows for a succession of two motions with n possibilities each, to produce n^2 distinct complex motions (n is the number of the FSMs). The FSMs that are integrated in the design are only those that led to the detection of the four simple motions. The complex motions are segmented to two simpler motions and the four FSMs (of the simple motions) are reused for each segment, as shown in Fig. 2.

In order to avoid false positives by successions of the same motion (e.g., "forward-forward"), these cases are not considered as complex motions, leading to $\mathrm{n}^2 - \mathrm{n}$ well-defined complex motions. *The maximum number of complex motions* that the system can detect is $4^2 - 4 = 12$. In addition, it recognizes the simple motions as such (e.g., we detect "forward" but do not look for "forward-forward"), leading to 16 well-defined motions. The vocabulary of motions that the system can detect is shown in Fig. 3.

In terms of hardware complexity, the FSMs are the subsystems that have the greatest requirements in resources. Consequently, the implementation of just four $(\mathrm{n} = 4)$ FSMs keeps

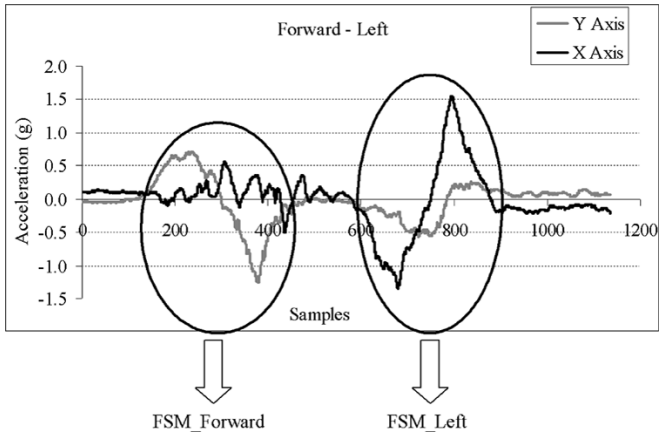

Fig. 2. Complex motion segmentation. A succession of simple motions ("forward" and "left") produces a complex one, whose recognition comprises of substeps. A simple motion is recognized during each step by the respective FSM.

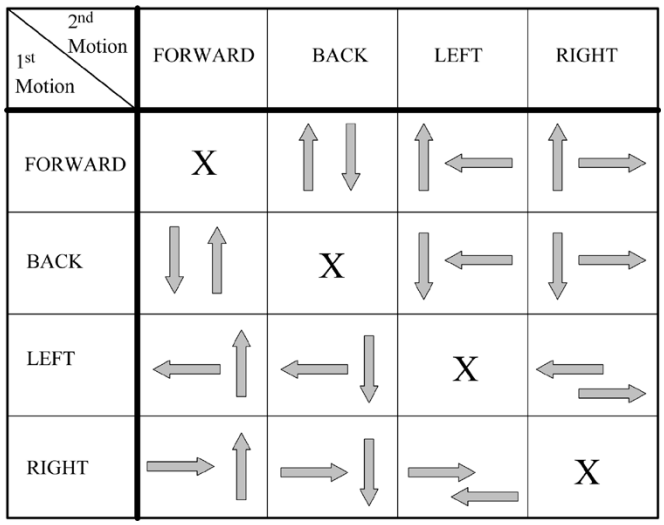

Fig. 3. Complex motion vocabulary. It comprises of four simple motions (forward, back, left, right) and twelve complex ones.

the hardware cost to O(n), even if the system is able to recognize $\mathrm{O}(\mathrm{n}^2)$ motions. Although there is an overhead on the control unit complexity, the latter is not of great importance and does not influence the overall system's hardware complexity. This approach is much more efficient compared to the idea of implementing one FSM per motion (complexity $\mathrm{O}(\mathrm{n}^2)$).

IV. System Adaptation to Individual Needs

It can be said that the general form of each simple motion is the same, regardless of the specific user. For example, a forward movement comprises of a set of accelerations, followed by samples of stable velocity, which are then followed by a set of decelerations (hand starts moving, then moves without accelerating and then stops). However, even if this procedure is similar for all users, it is not exactly the same. Clinical tests (of limited scope) have shown that the motions of an impaired person are not as even as the respective motions of an unimpaired one. Moreover, the speed of execution may vary even during the same motion. Thus, the system must have the capability to adapt to the different executions of the same motion by different users.

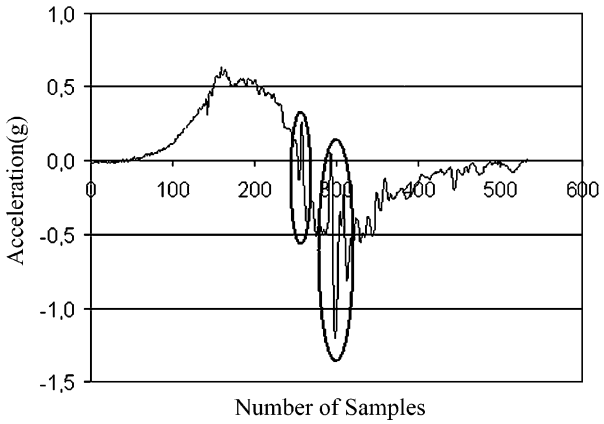

Fig. 4. Local extrema of a motion. Several factors (tremor, wrong movement, deviation from the calibration norm) may influence the smooth execution of a motion. Therefore, undesirable ripples usually appear on the acceleration data.

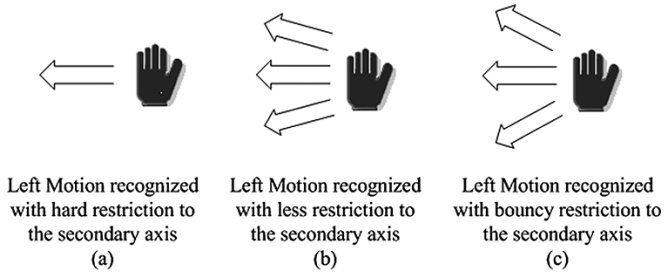

Fig. 5. Examples of a motion with various restrictions to its secondary axis. The more rigorous the restriction is, the less flexible in coping with errors during motion execution, the recognition system is. On the other hand, the more flexible the system is, the less effective in recognition is. (a) Left motion recognized with hard restriction to the secondary axis. (b) Left motion recognized with less restriction to the secondary axis. (c) Left motion recognized with bouncy restriction to the secondary axis.

The idea of adaptation primarily affects the FSMs that are responsible for motion recognition and is based on the determination of the thresholds that must change according to each user.

The procedure that has been implemented for the calculation of the thresholds is an unsupervised learning method. It comprises of two subprocesses: digital representation of the motion's acceleration sequence and calculation of the appropriate thresholds, based on the previous approximation.

A. Digital Representation of a Motion

The basic problem that we had to overcome was the local extrema that may appear during the execution of a motion, as it is shown in Fig. 4. The best way to address this problem is to ignore the sample sets that have such a behavior. In order to find them, we check whether the difference between the value of each sample and the value of the four previous ones is greater than a predefined acceleration value, which is considered as the physiological difference between adjacent samples.

The next step is the approximation of the accelerations in the graph. As the acceleration samples regard two axes (X, Y) the procedure could be repeated twice. However, experimental results showed that we need thresholds only for the *primary axis* of the motion (i.e. the axis in which the greatest mobility is observed). The other axis (called secondary) is characterized by less mobility than the primary one and we only need to designate upper and lower bounds for the acceleration, instead of thresholds. These bounds are set to -0.3 and $+0.3$ g. The reason for such a choice is presented in Fig. 5. A rigorous restriction on the secondary axis of a motion [Fig. 5(a)], leads to inability to cope with errors during the execution of a motion. On the contrary, a very loose restriction on the secondary axis [Fig. 5(c)] results in too many errors (false positives) during the execution of the motion. In this case, overlapping of the motions may take place. In order to avoid both these cases, we determined "middle-of-the-road" bounds for the secondary axis, that is, the case in which small motion sequence deviations are acceptable [Fig. 5(b)].

We find the primary axis of the motion, by checking the peak-to-peak distance of the two axes. Then, we approximate the acceleration graph of the primary axis. The basic concept is to divide the graph in windows of constant size and transfer the problem to the approximation of these windows. *Each of the windows can be (in terms of acceleration) incremental, decremental, or constant.*

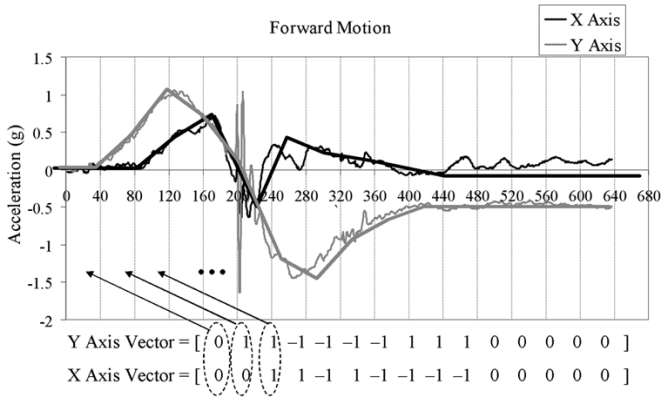

Fig. 6. Example of motion approximation. The motion sequence is divided in windows of a constant size equal to 40 samples The behavior of the acceleration data in each window determines the window type (whether it is incremental, decremental or constant). This information is stored in a vector (one per axis), using a simple coding (1 for incremental, -1 for decremental, and 0 for constant window).

The result of the approximation process is a vector, named st(), whose positions correspond to the respective windows, as shown in Fig. 6. The value of each position determines the respective window type (1 for incremental, -1 for decremental, 0 for constant).

B. Calculation of a Motion's Characteristic Thresholds

Threshold calculation comprises of two substeps. First, threshold positions are determined (i.e., we determine where in the sequence of acceleration data, thresholds are needed for this sequence to be described). Second, the values of these thresholds are calculated. We should point out that the current implementation uses a constant number of thresholds per motion (four thresholds). Minor changes can lead to an implementation with a variable threshold number. However, such an approach requires additional hardware resources which is not in agreement with our low-cost specifications.

In terms of threshold positions, thresholds are assigned to sequences that correspond to a set of accelerations followed by a set of decelerations. Such sequences occur during a hand movement. The algorithm that finds threshold positions is described

TABLE I
SEQUENCES OF WINDOWS THAT LEAD TO THRESHOLD POSITION DETECTION

State	Windows st(A)	st(B)	st(C)	st(D)	Action
0	1(-1)	-1(1)			Remove A (if inserted). State=3
0	1(-1)	1(-1)			Remove A (if inserted), Insert B. State=0
0	1(-1)	0			State=1
1	1(-1)	0	-1(1)		Insert A, Insert C (if not inserted). Insert Intermediate Threshold. State=0
1	1(-1)	0	1(-1)		Remove A (if inserted), Insert C. State=0
1	1(-1)	0	0		State=2
2	1(-1)	0	0	-1	Insert A, Insert D (if not inserted). Insert Intermediate Threshold. State=0
3	1(-1)	-1(1)	-1(1)		Insert A, Insert C (if not inserted). Insert Intermediate Threshold. State=0
3	1(-1)	-1(1)	0		Insert A, Insert B (if not inserted). Insert Intermediate Threshold. State=1
3	1(-1)	-1(1)	1(-1)		State=4
4	1(-1)	-1(1)	1(-1)	0	Insert C (if not inserted). State=0
4	1(-1)	-1(1)	1(-1)	1(-1)	Insert D (if not inserted). State=0

in Table I. It is actually an FSM that tracks the sequences of the windows (as they were previously determined) and assigns thresholds according to predefined templates.

The column "Windows" of Table I refers to the type of the windows that are currently checked [described by the vector st()]. Window A is always the last in a sequence (e.g., in a sequence of three windows C is the current window, B is the previous of C, and A is the previous of B). Predefined sequences and their corresponding actions are included in the table, e.g., a sequence of $\mathrm{st(A)} = 1, \mathrm{st(B)} = 0$ and $\mathrm{st(C)} = -1$ leads to the "action" described in the fourth row of the table, etc. Values included in parentheses of the column "Windows" describe complementary cases. The actions influence the contents of a vector named "threshpos."

Sometimes an action includes the insertion of an extra threshold, called "intermediate." This action is performed in special cases, when the duration of a movement is too small and less than four thresholds may be detected. When the scanning of all the sequences is completed, we calculate the thresholds' values, based on the vector "threshpos" and on the minimum and maximum values of each window.

V. UNSUPERVISED TRAINING

By using the PC as the user interface, we collect data for each simple motion from individuals. These data are the patterns, on which the adaptation procedure is based. After calculating the appropriate thresholds for each simple motion following the procedure that was previously described, we download them to a nonvolatile memory, in order to be available for further use. The AVR microcontroller offers an on-chip nonvolatile memory. The microcontroller loads the thresholds from this memory on power-up and sends them to the FPGA, where they are stored in registers. The FSMs have access to these registers and can read the respective thresholds. This way, they recognize the motions that are similar to the patterns that were provided by the user. *Such an approach leads to a reconfigurable system with capability to adapt to individual needs but without recompilation of the design*. The FPGA is initialized with a fixed design and with changeable threshold and delay data after it is configured.

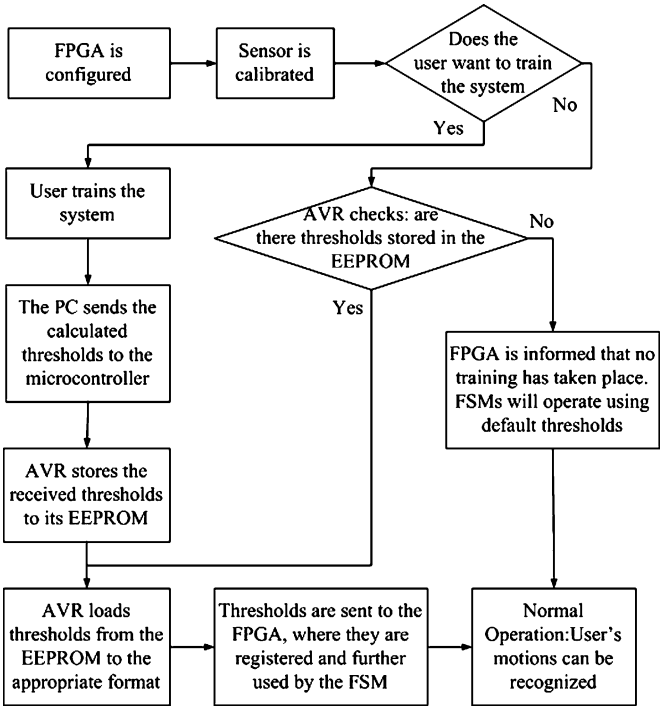

Fig. 7. The overall procedure the system follows until its normal operation. Sensor calibration follows the FPGA configuration. The user has then two choices: to train the system or to bypass this procedure and proceed to the normal operation. In the first case, the system enters the training mode and tells the user what to do.

It must be noted that the efficiency of the algorithm is increased when the training procedure is repeated more than once for each motion. The final thresholds result by averaging the respective thresholds of the repetitions. This way, potential variations during the execution of a motion are compensated.

The entire process, starting from power-up of the system, is shown in Fig. 7. If no training has taken place, the operation is based on default thresholds. The calculation of these fixed thresholds was based on experimental results.

VI. EXPERIMENTAL RESULTS AND MODEL VALIDATION

Many experiments have been performed in order to evaluate the behavior and performance of the system. The implementation of the FSMs in software allowed for substantial experimentation with computational models, thresholds, and delays. Over 150 experiments have been done with an unimpaired person (who will be called reference person), and over 100 experiments have been done with a kinetically challenged person. All experiments were recorded in a benchmark, which can be reused every time a design change is made in the system.

Fig. 8 presents examples of motion approximation and threshold calculation for motions executed by a reference and a kinetically challenged person. The horizontal axis represents the number of samples whereas the vertical axis represents acceleration, expressed in g (gravity's metric). As it is shown, the acceleration data time series for the same motion are similar for both persons, but the delays and thresholds are substantially different for different persons. The acceleration data series are of different lengths. This implies that speed of execution is different and, therefore, accelerations are different (e.g., when

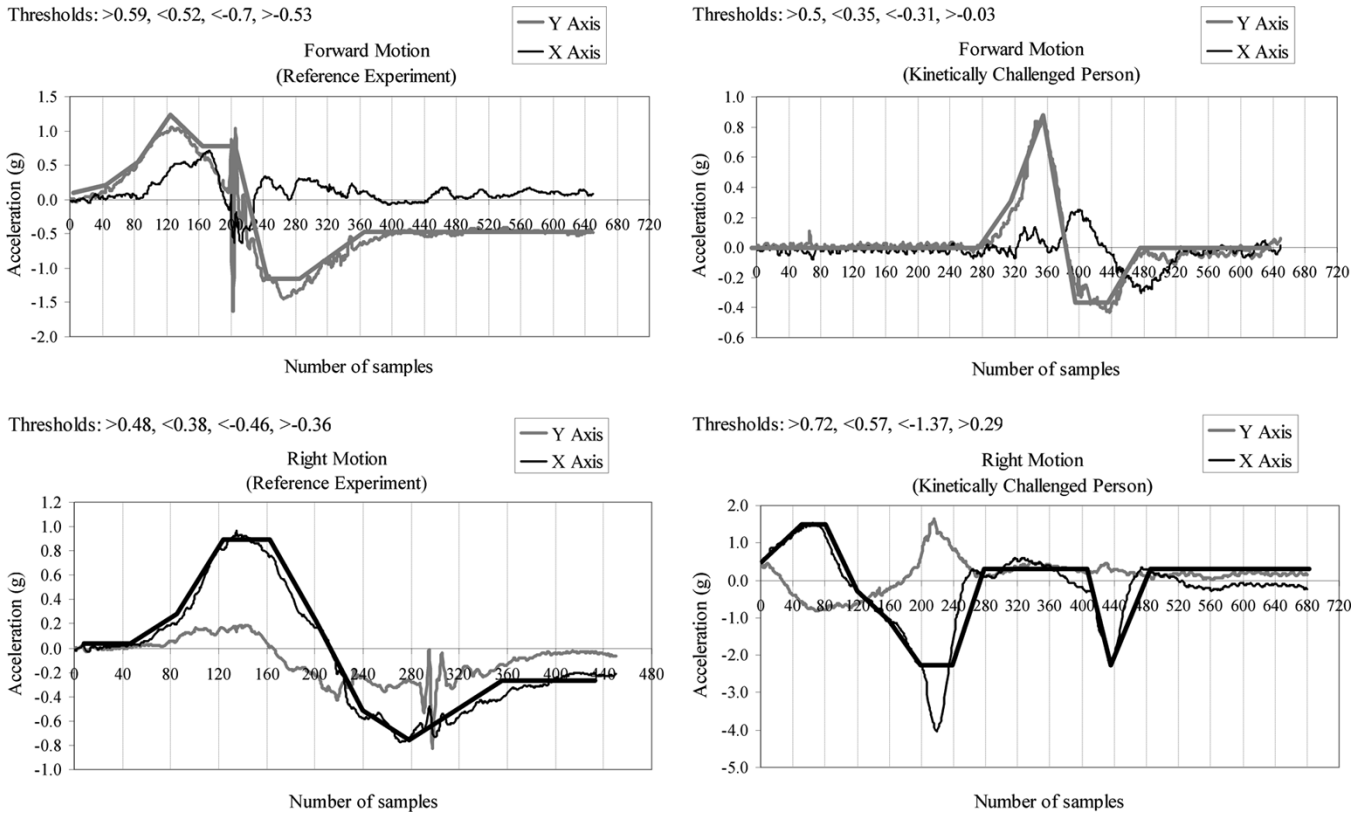

Fig. 8. Examples of motion approximation and threshold computation. Only the thresholds for the primary axis of each motion are presented here. The thresholds (computed by the approximation vectors of the motion) adapt to the different acceleration data of different persons. The efficiency of the training procedure is increased when it is repeated more than once for each motion.

TABLE II
RECOGNITION EFFECTIVENESS (PERCENTAGE OF CORRECTLY IDENTIFIED MOTIONS)

Simple motions	>95%
Forward + Back/Right/Left	95%
Back + Forward	95%
Right + Left	95%
Left + Right	95%
Back + Right/Left	90%
Right + Forward/Back	80%
Left + Forward/Back	80%

a motion is executed fast, the hand accelerates very rapidly). These differences impose the need for threshold adaptation to individual needs. The motion detection results were satisfactory in both cases. Especially, the efficiency of the algorithm was increased when the training procedure was repeated more than once for each motion.

One of the significant results of the experiments was that the preferred motions for the person with kinetic challenges are forward, back, left, right, something that led to the motion detection model. The motion detection model works very well for the (generally irregular) motions that a person with kinetic disorders can perform.

In Table II, the experimental results are presented. It concerns experiments with the reference person. It represents the recognition effectiveness of the scheme for the simple and complex motions. The model can recognize simple motions with better than 95% success. Regarding complex motions whose effectiveness is lower (80%), the change of the grade of calibration after the execution of the first segment of the motion contributes to this low percentage. Obviously, the model needs to be improved.

Something that needs to be clarified is that the training algorithm we implemented does not produce exactly the same thresholds even for similar motions. This is reasonable if we think that precisely replicated motions (in terms of the acceleration data) cannot be produced even from the same person. The maximum and minimum values will always be close, but never the same. Therefore, we are primarily concerned with acceleration ranges and not with absolute values (e.g. a threshold of 0.4 g will hardly make any difference as compared to a threshold of 0.45 g, but will be very different from a threshold of 0.6 g). This is illustrated in Fig. 8, where we can observe that the first threshold for the first three motions is quite the same (around 0.5 g), since the maximum acceleration value of all these motions is quite similar (around 1 g).

An unexpected result from this work was that the connection of our device to a PC as a mouse, offered to a kinetically challenged user a more convenient way of communicating with the PC versus the traditional mouse. An application that exploits the above observation and the ability of the accelerometer to measure tilt of the hand has been implemented. The user is able to

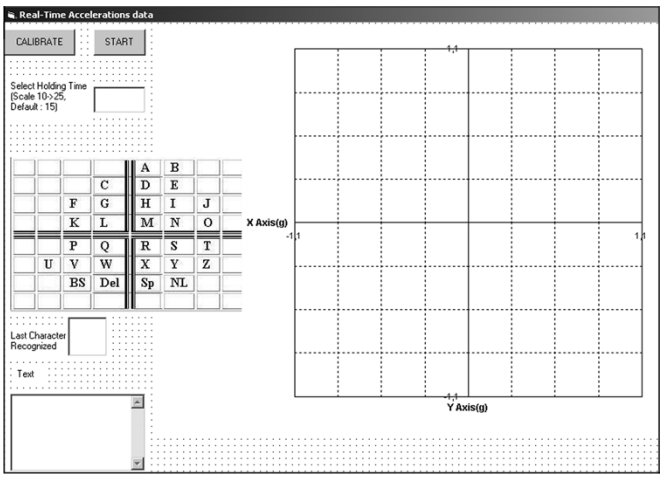

Fig. 9. Real time application for text input. Boxes of the big square on the right correspond to a different character. The mapping between characters and boxes is shown in the small square on the left, e.g., if the "cursor" is placed above the fifth box of the first row for a predefined time, then 'A' will be recognized.

input text to a PC by giving the appropriate tilt to his or her hand, on which the accelerometer is mounted. The underlying principle is that for each tilt of the hand (which subsequently remains firm), the acceleration of gravity has a unique and easily determined (X, Y) vector. This vector can be directly mapped on a grid.

The screen of the PC is segmented to several boxes, each of which corresponds to a character. The navigation in this environment is done through a "cursor," whose movement is controlled by the tilt of the user's hand. This tilt is measured by the accelerometer that sends the data to the software application, which directs the cursor to the appropriate area of the screen. The interface of this real-time application is shown in Fig. 9 and has been designed having in mind several ergonomic limitations.

VII. Conclusion and Future Work

We have presented a 2-D motion detection model, which has been implemented in reconfigurable hardware for a low-cost solution. A simple vocabulary of motions can be used to form more complex motions. The model is a good compromise cost/performance-wise. The system can be trained to individual users' needs and it can be extended to more motions, or more complex ones. The next step for the development of the system is extensive clinical testing, and deployment, whereas in the lab we need to work on improvement of the motion recognition accuracy.

Acknowledgment

The authors wish to thank Dr. T. Kean of Algotronix, Ltd. for his contributions and his valuable advice in the early stages of this project, as well as S. Koutsorinakis and M. Kimionis. They also acknowledge generous donations to their educational and research activities by Xilinx Corporation, and electronics parts support by Analog Devices.

References

[1] V. Kumar, T. Rahman, and V. Krovi, "Assistive devices for people with motor disabilities," in *Encyclopedia of Electrical and Electronics Engineering* New York, 1997.

[2] C. Lau and S. O'Leary, "Comparison of computer input devices for persons with severe physical disabilities," *Am. J. Occupat. Ther.*, vol. 47, no. 11, Nov. 1993.

[3] L. M. Bergasa, M. Mazo, A. Gardel, J. C. García, A. Ortuno, and A. E. Mendez, "Guidance of a wheelchair for handicapped people by head movements," in *Proc. 7th Int. Conf. Emerging Technol. Factory Automation (ETFA' 99)*, Barcelona, Spain, Oct. 1999, pp. 105–111.

[4] L. Dipietro, A. Sabatini, and P. Dario, "Evaluation of an instrumented glove for hand-movement acquisition," *J. Rehabil. Res. Develop.*, vol. 40, no. 2, pp. 179–190, Mar.–Apr. 2003.

[5] L. Bretzner, I. Laptev, T. Lindeberg, S. Lenman, and Y. Sundblad, "A Prototype System for Computer Vision Based Human Computer Interaction,", Tech. Rep. ISRN KTH/NA/P-01/09-SE, Apr. 2001.

[6] S. Miyazaki, A. Ishida, and A. Komatsuzaki, "A clinically oriented video-based system for quantification of eyelid movements," *IEEE Trans. Biomed. Eng.*, vol. 47, no. 8, pp. 1088–1096, Aug. 2000.

[7] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, Jun. 2002.

[8] K. Nakahara, D. L. Jaffe, and E. E. Sabelman, "Development of a second generation wearable accelerometric motion analysis system," in *Proc. 2nd VA National Department of Veterans Affairs Rehabilitation Research and Development, "The Next Generation"*, Arlington, VA, Feb. 20–22, 2000.

[9] R.-X. Chen and L.-G. Chen, "Design and implementation of FPGA wheelchair controller," in *Proc. 10th VLSI Design/CAD Symp.*, Nantou, Taiwan, Aug. 18–21, 1999, pp. 57–60.

[10] M. Mazo *et al.*, "An integral system for assisted mobility," *IEEE Robot. Automat. Mag.*, vol. 8, no. 1, pp. 46–56, Mar. 2001.

[11] S. S. Smith, "Measurement of neuromuscular performance capacities," in *The Biomedical Engineering Handbook*, 2 ed, J. D. Bronzino, Ed. Baltimore, MD: Lippincott Williams & Wilkins, 2000, vol. II, pp. 148-1–148-19.

[12] A. Dollas, K. Papademetriou, N. Aslanides, and T. Kean, "A reconfigurable embedded input device for kinetically challenged persons," in *Proc. Eur. Conf. Field Programmable Logic (FPL)*, Belfast, N. Ireland, Aug. 2001, pp. 326–335.

[13] K. Papademetriou, A. Dollas, and S. Sotiropoulos, "A second generation embedded reconfigurable input device for kinetically challenged persons," in *Proc. IEEE Symp. FPGA's for Custom Computing Machines (FCCM)*, Napa Valley, CA, 2003, pp. 294–295.

Apostolos Dollas (M'84–SM'92) received the B.S., M.S., and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign in 1982, 1984, and 1987, respectively.

He is Professor of Electronic and Computer Engineering, Technical University of Crete, Chania, Greece, where he has served one term as Department Chairman. During 1986–1994, he has been on the faculty of the Electrical Engineering and of Computer Science, Duke University. He is the Director of the Microprocessor and Hardware Laboratory. He conducts research and teaches in the areas of reconfigurable computing, rapid system prototyping, embedded systems, and application specific high-performance digital systems. In all of these areas, he places emphasis on the development of fully functional prototypes.

Prof. Dollas is a member of the IEEE Computer Society. He belongs to the Eta Kappa Nu and Tau Beta Pi and has received the IEEE Computer Society Golden Core Member award, the IEEE Computer Society Meritorious Service Award, and twice the Department of Computer Science Award for Outstanding Teaching at the University of Illinois at Urbana-Champaign. He has served on the program committee of several IEEE conferences and workshops, including FCCM (since 1993), RSP (since 1990, Program co-chairman in 2001, general co-chairman in 2004) TAI (1989–1991), WASP (2002), and ACM WBMA (2003).

Stamatios Sotiropoulos received the B.S. degree in electronic and computer engineering from the Technical University of Crete, Chania, Greece, in 2003. He received the M.S. in biomedical engineering from the University of Minnesota at Twin-Cities in 2005. He is currently working towards the Ph.D. degree in bioengineering at Arizona State University, Tempe.

His current research interests are in neuroengineering and brain stimulation.

Mr. Sotiropoulos is a student member of the Society for Neuroscience.

Kyprianos Papademetriou received the Diploma and M.Sc. degrees in electronic and computer engineering from the Technical University of Crete, Chania, Greece, in 1998 and 2003, respectively. He is currently working towards the Ph.D. degree in the Department of Electronic and Computer Engineering, Technical University of Crete.

During 1998–1999, he was with the R&D Department of ATMEL S.A, where he worked on hardware implementation of high-speed telecommunication protocols. His current research interests are in embedded systems and reconfigurable computing.