J Real-Time Image Proc (2008) 3:163–176
DOI 10.1007/s11554-008-0073-1

SPECIAL ISSUE

Real-time human action recognition on an embedded, reconfigurable video processing architecture

Hongying Meng · Michael Freeman · Nick Pears · Chris Bailey

Received: 13 July 2007 / Accepted: 17 January 2008 / Published online: 14 February 2008

Abstract In recent years, automatic human action recognition has been widely researched within the computer vision and image processing communities. Here we propose a real-time, embedded vision solution for human action recognition, implemented on an FPGA-based ubiquitous device. There are three main contributions in this paper. Firstly, we have developed a fast human action recognition system with simple motion features and a linear support vector machine classifier. The method has been tested on a large, public human action dataset and achieved competitive performance for the temporal template class of approaches, which include "Motion History Image" based techniques. Secondly, we have developed a reconfigurable, FPGA based video processing architecture. One advantage of this architecture is that the system processing performance can be reconfigured for a particular application, with the addition of new or replicated processing cores. Finally, we have successfully implemented a human action recognition system on this reconfigurable architecture. With a small number of human actions (hand gestures), this stand-alone system is operating reliably at 12 frames/s, with an 80% average recognition rate using limited training data. This type of system has applications in security systems, man–machine communications and intelligent environments.

Keywords Human motion recognition · Reconfigurable architectures · Embedded computer vision · FPGA · Machine learning

H. Meng (✉) · M. Freeman · N. Pears · C. Bailey
Department of Computer Science,
University of York, Heslington,
York YO10 5DD, UK
e-mail: menghongying@tsinghua.org.cn

1 Introduction

Ambient intelligence (AmI) reflects an emerging and popular field of research and development that is oriented towards the goal of "intelligent" or "smart" environments that react in an attentive, adaptive, and active way to the presence and activities of humans and objects in order to provide smart services to the inhabitants of these environments.

An environment is said to be "perceptive" when it is capable of recognizing and describing things, people and activities within its volume. Input can be obtained from sensors for sound, images, and haptics. For example, video capture is low cost, widespread, and can be used for monitoring human events.

Event recognition is an important goal for building intelligent systems which can react to what is happening in a scene. It is also a fundamental building block for interactive systems which can respond to gestural commands, instruct and correct a user learning athletics, gymnastics or dance movements, or interact with live actors in an augmented dance or theatre performance.

Recognizing motions or actions of human actors from image sequences is also an important topic in computer vision with many fundamental applications in video surveillance, video indexing and social sciences. Event detection in video is becoming an increasingly important application for computer vision, particular in the context of activity recognition [1].

Model based methods are extremely challenging as there is a large degree of variability in human behaviour. The highly articulated nature of the body leads to high dimensional models and the problem is further complicated by the non-rigid behaviour of clothing. Computationally intensive methods are needed for non-linear modelling and

optimization. Recent research into anthropology has revealed that body dynamics are far more complicated than was earlier thought, affected by age, ethnicity, gender and many other circumstances [11].

Appearance-based models are based on the extraction of a 2D shape model directly from the images, to be classified (or matched) against a trained one. Motion-based models do not rely on static models of the person, but on human motion characteristics. Motion feature extraction and selection are two of the key components in these kinds of human action recognition systems.

In this paper, we propose a human motion recognition system that is both fast and accurate. It is designed for applications in security systems, man–machine communication, and other cases of AmI. It is implemented on our field-programmable gate array (FPGA) based reconfigurable video processing architecture, which we call "Videoware". Experimental results demonstrate the efficiency and reliability of the system.

The rest of this paper is organized as follows: in Sect. 2, we give an introduction to our human action recognition system and we evaluate the performance of the system on a challenging, large, public human action dataset. In Sect. 3, we introduce the reconfigurable video processing architecture. In Sect. 4, we introduce the implementation of the human action recognition on the reconfigurable architecture and give some experimental results for this real-time embedded vision system. Finally, we present some discussion and the conclusions.

2 Human motion recognition system

2.1 Related works on human action recognition

Aggarwal and Cai [1] present an excellent overview of human motion analysis. Of the appearance based methods, template matching has gained increasing interest recently. Moeslund et al. [20] have produced a review paper for the most recent advances.

Bobick and Davis [7] pioneered the idea of temporal templates. They use motion energy images (MEI) and motion history images (MHI) to recognize many types of aerobics exercise. Bradski and Davis [8] proposed the motion gradient orientation (MGO) to explicitly encode changes in an image introduced by motion events.

Davis [10] presented a useful hierarchical extension for computing a local motion field from the original MHI representation. The MHI was transformed into an image pyramid, permitting efficient fixed-size gradient masks to be convolved at all levels of the pyramid, thus extracting motion information at a wide range of speeds. The hierarchical MHI approach remains a computationally inexpensive algorithm to represent, characterize, and recognize human motion in video.

Ogata et al. [21] proposed modified motion history images (MMHI) and used an eigenspace technique to realize high-speed recognition. The experiment was performed on recognizing six human motions and the results showed satisfactory performance of the technique.

We note that, in some of these methods [9, 6, 14, 22, 25, 28], the motion features employed are relatively complex, which implies significant computational cost on building the features. Some methods [6, 7, 8, 10, 21, 30, 31] require segmentation, tracking or other prohibitive computational cost, that is not suitable for real-time embedded vision applications.

To our knowledge, there are no publications on the implementation of visual human action recognition using an FPGA platform, which is the main theme of this paper.

2.2 MHI/SVM based recognition system

We now propose a novel solution for fast human action recognition. This has been reported in our previous work [16]. In this approach, a linear support vector machine (SVM) was chosen as the classifier and the MHI provided our fundamental features.

There are three reasons for choosing a linear SVM as the classifier in the system. Firstly, the SVM is a classifier that has achieved excellent performance in many real-world classification problems. Secondly, the SVM can deal with very high dimensional feature vectors, which means that there is plenty of freedom to choose the feature vectors. Finally the classifier is able to operate very quickly during the recognition process.

A normal recognition system includes two parts: a learning (or training) part and a classification part. These two parts of our recognition system are showed separately in Fig. 1.

The feature vectors are to be obtained using motion information directly from the input video. It is expected that the feature extraction algorithms and dimension reduction algorithms should be as simple as possible to enable hardware based implementation.

The learning part is processed using video data collected off-line. After that, the computed parameters for the classifier can be used in a small, embedded computing device such as a FPGA or digital signal processor (DSP) based system, which can be embedded in the application and can give real-time performance.

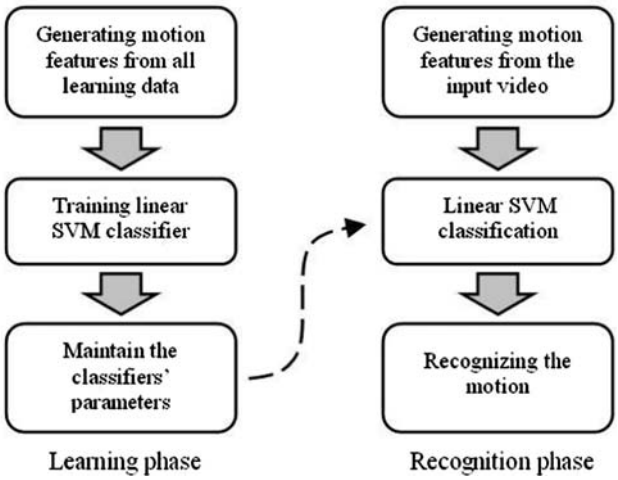

Fig. 1 SVM based human action recognition system. In the learning part, the motion feature vector was used for training a SVM classifier, and the parameters computed were used in the recognition part

2.3 Motion features

The recording of human actions usually needs very large amounts of digital storage space and it is time consuming to browse the whole video to find the required information. It is also difficult to deal with this huge amount of data in detection and recognition. Therefore, several motion features have been proposed to compact the whole motion sequence into one image to represent the motion. The most popular ones are the MHI [7], MMHI [21] and MGO [8]. These three motion features have the same size as the frame of the video, but they maintain the motion information within them. In our experiments, it has been found that the MHI representation gives a system with a better classification performance than the other two features [16].

An MHI is a kind of temporal template. It is the weighted sum of past successive images and the weights decay as time lapses. Therefore, an MHI image contains past raw images within itself, where most recent image is brighter than past ones.

Normally, an MHI $H_\tau(u,v,k)$ at time k and location (u,v) is defined by the following Eq. 1:

$$H_\tau(u,v,k) = \begin{cases} \tau & \text{if } D(u,v,k) = 1 \\ \max\{0, H_\tau(u,v,k-1) - 1\}, & \text{otherwise} \end{cases} \quad (1)$$

where $D(u,v,k)$ is a binary image obtained from subtraction of frames, and τ is the maximum duration a motion is stored. In general, τ is chosen as constant 255 where MHI can be easily represented as a greyscale image. An MHI pixel can have a range of values, whereas the MEI is its binary version. This can easily be computed by thresholding $H_\tau > 0$.

Figure 2 shows the motion features of MHI (b) and MMHI (c) of a bird flight in the sky (a). From these features, we can clearly determine how the bird flew in the sky even though we did not see the original video clip, since these features retain the motion information within them.

From these motion images, some researchers have attempted to extract further low-dimensional feature vectors. In [7], Hu moments, which are good at expressing shape discrimination, were used to extract the motion feature from the MHI. However, in order to keep the simplicity for hardware implementation, we use the simplest method to transform a motion feature image (e.g. MHI) into a plain vector, based on the pixel scan order (pixel by pixel) to feed a SVM classifier.

2.4 Support vector machine

SVM is a state-of-the-art classification technique with large application in a range of fields including text classification, face recognition and genomic classification, where patterns can be described by a finite set of characteristic features.

We use the SVM for the classification component of our system. This is due to SVM being a classifier that has excellent performance on many real-world classification problems. Using arbitrary positive definite kernels provides a possibility to extend the SVM capability to handle high dimensional feature spaces.

Originally, the SVM is a binary classifier in a higher dimensional space where a maximal separating hyperplane is constructed. Two parallel hyperplanes are constructed on each side of the hyperplane that separates the data. The separating hyperplane is the hyperplane that maximizes the distance between the two parallel hyperplanes. If we have a training dataset $\{\mathbf{x}_i | \mathbf{x}_i \in R^d\}$, and its binary labels are denoted as $\{y_i | y_i = \pm 1\}$, the norm-2 soft-margin SVM can be represented as a constrained optimization problem

Fig. 2 In this video sample, a bird flies in the sky (*left*). The features MHI (*middle*) and MMHI (*right*) both have retained the motion information of the bird

$$\min_{w,b,\xi} \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i \tag{2}$$

s.t.

$$\langle \mathbf{x}_i, \mathbf{w}\rangle + b \geq 1 - \xi_i, \quad y_i = 1,$$
$$\langle \mathbf{x}_i, \mathbf{w}\rangle + b \leq -1 + \xi_i, \quad y_i = -1,$$
$$\xi_i \geq 0,$$

where C is a penalty parameter and ξ_i are slack variables. The vector $\mathbf{w} \in R^d$ points perpendicular to the separating hyperplane. Adding the offset parameter b allows us to increase the margin. It can be converted by applying Lagrange multipliers into its Wolfe dual problem and can be solved by quadratic programming methods.

The primal optimum solution for weight vector $\mathbf{w}$ can be represented as

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i. \tag{3}$$

where $0 \leq \alpha_i \leq C$. Obviously, $\mathbf{w}$ can be expressed as a linear combination of the support vectors for which $\alpha_i > 0$. For a testing feature vector $\mathbf{x}$, the decision function η and its estimated label h are:

$$h(\mathbf{x}) = \text{sign}(\eta(\mathbf{x})) = \text{sign}(\langle \mathbf{w}, \mathbf{x}\rangle + b). \tag{4}$$

The original optimal hyperplane algorithm was a linear classifier. However, many researchers have created non-linear classifiers by applying a kernel trick [2] and thus the SVM can be generalized to the case where the decision function is a non-linear function of the data.

Multiclass SVMs are usually implemented by combining several two-class SVMs. In each binary SVM, only one class is labelled as "1" and the others labelled as "−1". The one-versus-all method uses a winner-takes-all strategy.

If there are M classes, then the SVM method will construct M binary classifiers by learning. During the testing process, each classifier will get a confidence coefficient $\{\eta_j(\mathbf{x})|j = 1,2,...,M\}$ and the class k with the maximum confidence coefficient will be assigned to this sample $\mathbf{x}$.

$$h(\mathbf{x}) = k, \quad \text{if } \eta_k(\mathbf{x}) = \max_{j=1}^{M}\left(\eta_j(\mathbf{x})\right). \tag{5}$$

Our human action recognition problem here is a multiclass classification case. If, for example, we have six classes, then six SVM classifiers are trained based on motion features such as the MHI obtained from human action video clips in a training dataset. For each SVM training, one class is labelled as "1" and the rest classes are labelled as "−1". After the training, each SVM classifier is represented by two parameters $\mathbf{w}$ and b. These parameters will be stored in the internal memory of the FPGA. In the recognition process, one inner product between obtained MHI and $\mathbf{w}$ will be calculated and added to b for each SVM classifier. Then the final predicted label for the action video will go to the class with the maximum one in the computed six values.

2.5 Performance evaluation

For the evaluation, we use a challenging human action recognition database recorded by Schuldt et al. [25]. It contains six types of human actions (walking, jogging, running, boxing, hand waving and hand clapping) performed several times by 25 subjects in four different scenarios: outdoors (s1), outdoors with scale variation (s2), outdoors with different clothes (s3) and indoors (s4).

This database contains 2,391 sequences. All sequences were taken over homogeneous backgrounds with a static camera with frame rate of 25 frames/s. The sequences were downsampled to the spatial resolution of 160 x 120 pixels and have a length of 4 s on average. To the best of our knowledge, this is the largest video database with sequences of human actions taken over different scenarios. All sequences were divided with respect to the subjects into a training set (eight persons), a validation set (eight persons) and a test set (nine persons).

Figure 3 shows examples of each type of human action in this dataset and their associate MHI motion features. In order to compare our results with those in papers [14, 25], we use the exact same training set and testing set in our experiments. The only difference is that we did not use the validation dataset in training. In the same manner as paper [14], each sequence is treated individually during the training and classification process.

In our system, the SVM was trained based on features obtained from human action video clips in a training dataset. Generally, we can have several types of actions in a video dataset. These video clips have their own labels such as "walking", "running" and so on. In classification, we actually have a six-class classification problem. At first, we create six binary SVM classifiers and each of them is related to one of the six classes. For example, there is one SVM classifier related to the class "walking", one for "jogging" and so on. In the training dataset, the video clips with label "walking" will have a label "1" in the "walking" SVM classifier while video clips belonging to all other classes have a label "−1". Secondly, we train these SVM classifiers on the training dataset. The SVM training can be implemented using programs freely available on the web, such as SVM$^{\text{light}}$ by Joachims [13]. Finally, we obtained several SVM classifiers with their associated parameters.

Our experiments are carried out on the all four different scenarios: outdoors, outdoors with scale variation, outdoors with different clothes and indoors. In the same manner as

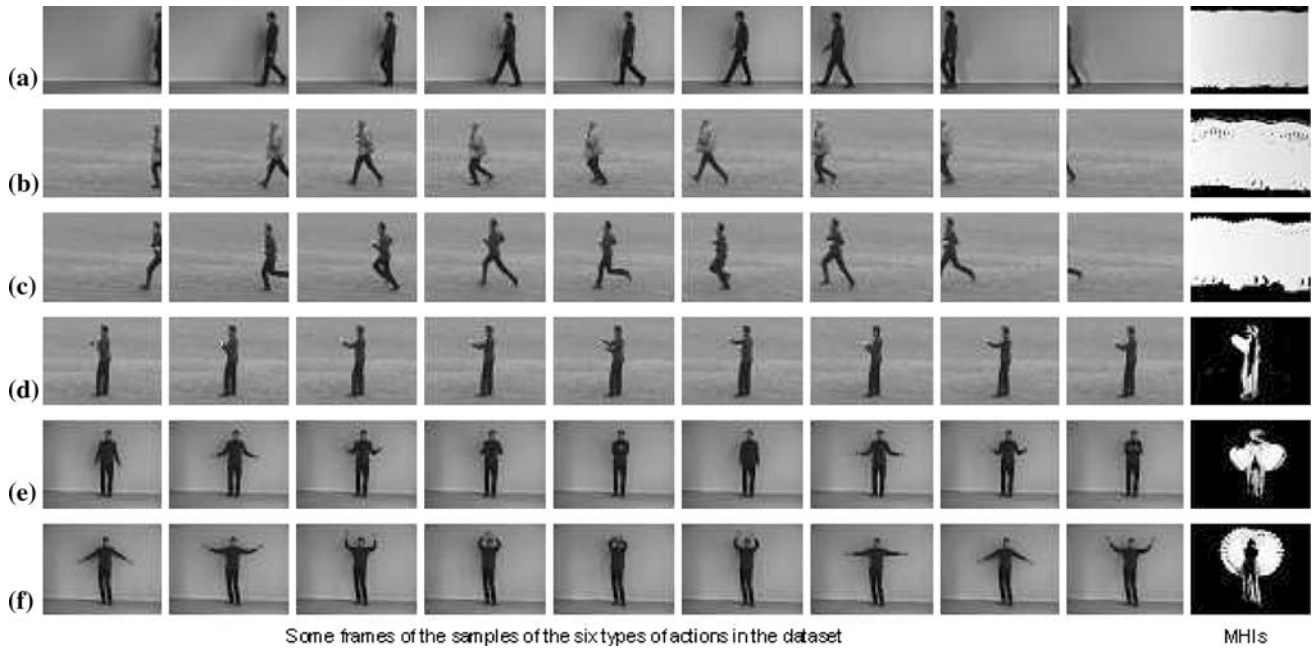

Fig. 3 Six types of human actions in the database: **a** walking, **b** jogging, **c** running, **d** boxing, **e** handclapping and **f** handwaving. The frames of the actions and their associate MHI features

Table 1 Ke's confusion matrix, trace = 377.8, mean = 63%

	Walk	Jog	Run	Box	Clap	Wave
Walk	**80.6**	11.1	8.3	0.0	0.0	0.0
Jog	30.6	**36.2**	33.3	0.0	0.0	0.0
Run	2.8	25.0	**44.4**	0.0	27.8	0.0
Box	0.0	2.8	11.1	**69.4**	11.1	5.6
Clap	0.0	0.0	5.6	36.1	**55.6**	2.8
Wave	0.0	5.6	0.0	2.8	0.0	**91.7**

Table 2 MHI's confusion matrix, trace = 381.2, mean = 63.5%

	Walk	Jog	Run	Box	Clap	Wave
Walk	**53.5**	27.1	16.7	0.0	0.0	2.8
Jog	46.5	**34.7**	16.7	0.7	0.0	1.4
Run	34.7	28.5	**36.1**	0.0	0.0	0.7
Box	0.0	0.0	0.0	**88.8**	2.8	8.4
Clap	0.0	0.0	0.0	7.6	**87.5**	4.9
Wave	0.0	0.0	0.0	8.3	11.1	**80.6**

paper (8), each sequence is treated individually during the training and classification process.

For the whole dataset, the classification confusion matrix is a good measure for the overall performance in this multiclass classification problem. Table 1 shows the classification confusion matrix based on the method proposed in paper [14]. Table 2 shows the confusion matrix obtained by our system, which uses the MHI. The confusion matrices show the motion label (vertical) versus the classification results (horizontal). Each cell (i,j) in the table shows the percentage of class i action being recognized as class j. Thus the main diagonal of the matrices show the percentage of correctly recognized actions, while the remaining cells show the percentages of misclassification. The mean recognition rate for our system is 63.5%, which is very similar to Ke's result of 63%.

From these two tables, we can see that in this six classes human action classification problem, our method did very well in distinguishing the last three classes "boxing", "handclaping" and "handwaving". But it is not effective in distinguishing the first three classes "walking", "jogging" and "running". The reason is that this dataset is really a challenging dataset. Some actions in these three classes "walking", "jogging" and "running" are very difficult to classify even by a human observer, for example, some subjects "jogging" actions are even slower other object's "running" action. However, in comparison with Ke's method, we use a simple MHI rather than volumetric features in which the dimension of feature vector might be a billion and our performance is a little bit better on this dataset.

3 Reconfigurable video processing architecture

Developments in AmI and ubiquitous computing have lead to the concept of disappearing computing [29], with a user being unaware that they are interacting with a collection of

computing nodes. Such devices have been termed context aware applications [24], smart devices that sense the real world they are operating in and use this information combined with a set of rules to enhance their operation. To continue to expand the functionality available from such devices, improved sensor technologies must be incorporated into these systems. The Amadeus Videoware project [23] aims to develop a hardware video processing architecture, which will support visually driven human interaction with a wide range of ubiquitous devices. These objectives require that computing technology is seamlessly integrated into an environment. This has become a reality with the ever decreasing cost, size and power requirements of embedded processors. However, an alternative approach to this traditional processor based solution is considered in this research. Instead of using a general purpose processor, a more hardware-based solution is taken, with the development of application specific intellectual property cores (IP cores) for FPGA or application-specific integrated circuit (ASIC) devices that can be optimized for the desired application. The main aim of this approach is to minimize power requirements and component costs by designing system on a chip (SOC) based systems. These reductions in power requirements and production costs are two of the main driving forces in current electronic system design, illustrated by the move away from board level design [with standard components, ASICs and application-specific standard products (ASSPs)] to SOC architectures. This trend has continued within the FPGA and ASIC design flows. Following Moore's law, the ever increasing amounts of resources available within these devices has allowed the designer to move away from "simple" SOC designs to multiprocessor SOC (MPSOC) and network on a chip (NOC) designs as illustrated in Fig. 4.

The traditional SOC design is based around a top level processor core, controlling the allocation of processing tasks to a set of data processing, hardware accelerators. These IP-cores are typically hardwired, i.e. control structures are implemented as hard coded state machines with highly pipelined or replicated data paths to improve processing performance. The resulting IP-core can obtain very high levels of processing performance, however, they tend to be inflexible, being optimized for a specific application. The development of these IP-cores is very labour intensive, taking a significant amount of time to design and debug when compared to a software implementation. With the increasing amounts of silicon area available to the designer, general purpose and application specific processor cores are replacing these hardware accelerator IP-cores, leading to the development of MPSOC and NOC designs. The type and number of processor cores chosen is highly dependent on the application's real time processing performance requirements, however, these can include RISC [5, 19], DSP or configurable processor cores [4, 27]. The per-unit processing performance of such systems when compared to a purely hardware implementation will of course be lower, however, the key advantage of this type of system is its flexibility, allowing a single design to be reconfigured for different applications via firmware modifications. Therefore, modern SOC designs will typically include multiple processor cores, each being allocated one or more tasks to improve parallelism within the system. Those functions requiring very high levels of processing performance which cannot be achieved by a software based implementation will of course still need to be implemented as hardware accelerator IP-cores, i.e. to allow the system to meet its real time processing deadlines.

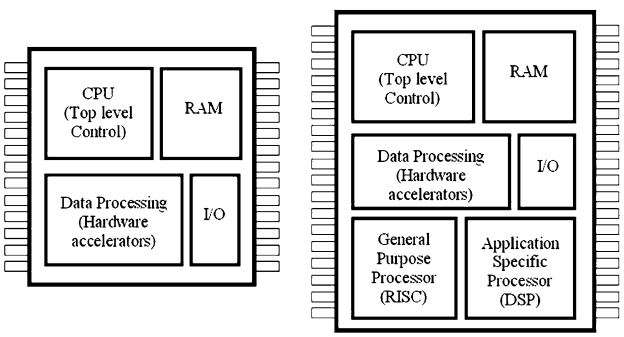

Fig. 4 Simple SOC (*left*) and MPSOC designs (*right*)

A central aim of the Videoware project is to implement a video component library (VCL) of generic image processing, computer vision and pattern recognition algorithms in an FPGA based architecture. The low level, high bandwidth processes, such as smoothing and feature extraction, will be implemented as hardware IP-cores, whilst higher level, lower bandwidth processes, such as task-oriented combination of visual cues, is implemented in a software architecture as shown schematically in Fig. 5. The advantage of this modular approach is that a system's processing performance can be reconfigured for a particular application, with the addition of new or replicated processing cores. This being simplified by using a MPSOC design with only those functions required for low level hardware interfacing or high levels of processing performance being implemented purely in hardware.

The hardware architecture shown in Fig. 5 has been implemented on a custom made FPGA board, the Amadeus ubiquitous system environment (USE) board [3]. This board is based on a Xilinx Spartan-III device [34], with 2 MB of external RAM and 8 MB of external ROM (this memory is also used to configure the FPGA via a complex programmable logic device (CPLD) configuration engine). The FPGA size can be selected to match a system's requirements, the board accepting three alternative devices: XC3S1500 (1.5M gates), XC3S2000 (2M gates) and

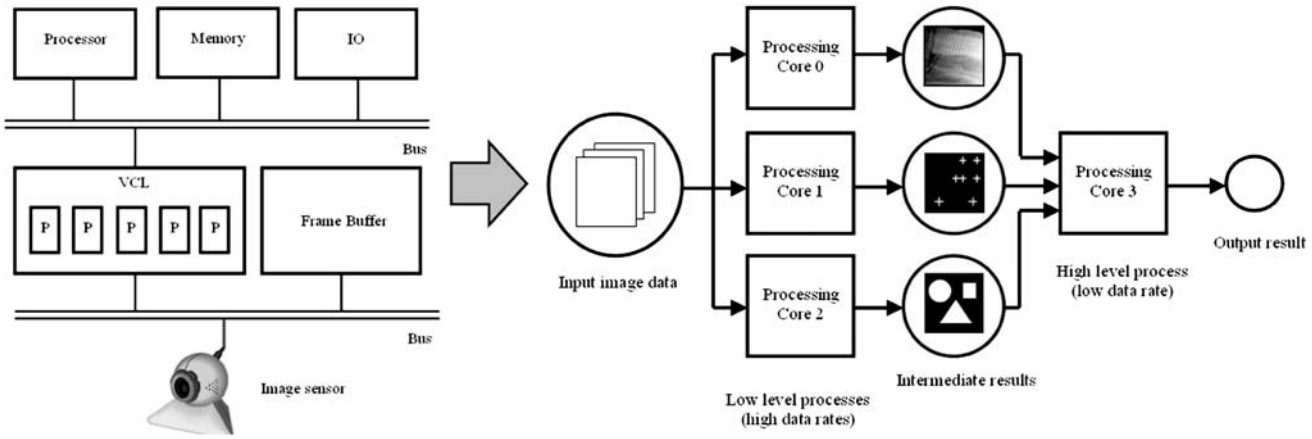

Fig. 5 Videoware processing architecture, VCL configured to form a virtual processing pipeline

XC3S4000 (4M gates). In addition to this a number of interface boards have also been developed to allow the easy connection of a camera [15], communications interfaces, e.g. LEDs, RS232 and additional external memory modules, e.g. SDRAM and SRAM.

4 Implementation of human action recognition on the reconfigurable architecture

4.1 System design and implementation

To minimize development time, i.e. the number of different hardware components that need to be developed, processor instruction sets and software developments tools that need to be learnt, each processing core is based on the same processor architecture. For the Xilinx Spartan-III device this means selecting from the Xilinx PicoBlaze (8 bit) [32], Xilinx MicroBlaze (32 bit) [33] or a third party processor IP-Core (not considered due to size constraints). When selecting a processor core, the data width, memory architecture and instruction set need to be matched to the application. In this case the majority of data sources use either an unsigned 8bit integer value, i.e. grey scale images, or a signed 8.8 fixed point representation (8 bit integer, 8 bit fraction). A 32 bit processor would require significant increases in both memory and logic. Therefore, it was decided that the 8 bit Xilinx PicoBlaze processor would be used. This decision does reduce the processing performance of these cores, e.g. operations requiring 16 or 32 bit operands require multiple instructions. The PicoBlaze processor is also not designed to be a raw "number cruncher" with an average performance of approximately 50 MIPS. Therefore, to compensate for this, a co-processor interface was added to the system bus allowing identified software bottlenecks to be moved into hardware, e.g. signed 8.8 fixed point multiplication, signed 24.8 fixed point accumulator, etc. In addition to this, operand pointer management has been moved out of software into dedicated hardware within the processor-to-wishbone bridge. This functionality was identified from the convolution operators required in a number of the video processing algorithms. Read and write pointers are now implemented in hardware being automatically updated when a value has been fetched from or written to memory. The processor can request either a single or block of data from a base address with a selected offset, greatly simplifying code structure and reducing code size. In general a software biased design approach was taken when developing each processing core, i.e. a co-processor was only added if a software based implementation does not achieve the required processing performance.

The block diagram of the FPGA motion recognition system is shown in Fig. 6. Each functional unit being implemented as hardware components written in VHDL. Control and IO tasks are centralized in the top level processor module. This module contains:

- PicoBlaze processor: top level control software, on boot-up initializes camera interface, etc., synchronizing and allocating tasks to processing cores to fulfil the system's processing requirements.
- Look up table ROM: configuration data used to initialize the system, e.g. camera control register values, frame size, etc.
- Scratch pad RAM: temporary storage for intermediate results. The inclusion of this memory is dependent on the variant of PicoBlaze processor used, i.e. the PicoBlaze KCPSM3 includes 64 B of internal scratch pad memory within its architecture.
- PicoBlaze to Wishbone bridge: the PicoBlaze processor does not support wait states, i.e. delayed read or write operations to memory, therefore, a bridge is required to interface the PicoBlaze to the Wishbone system bus

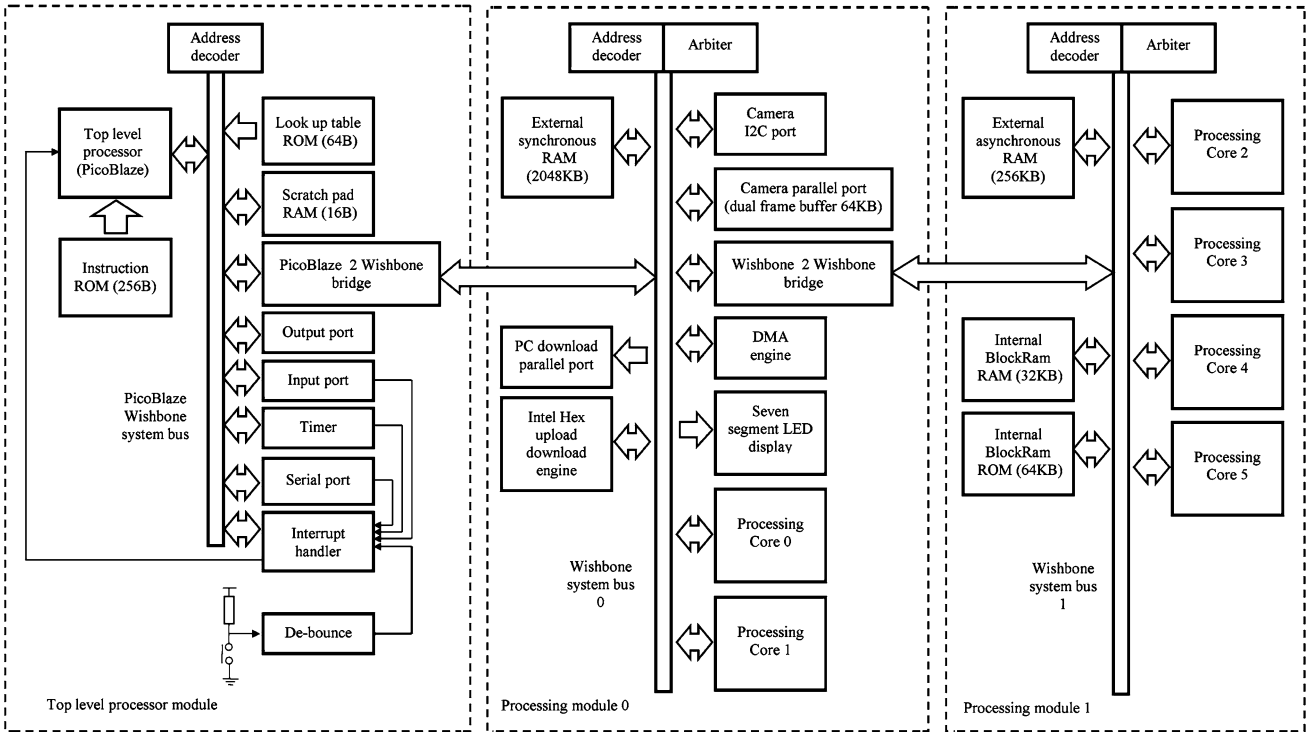

Fig. 6 System block diagram

[26]. The processor requests read or write operations from the bridge, data being stored in internal FIFO buffers.

- Input port: simple push buttons used to start system operations, DIP switch bank selects what configuration data should be used, e.g. camera sensitivity, etc.
- Serial port: displays result and debug information on PC based serial terminal.
- Interrupt handler: controls servicing of multiple interrupts.

The functionality required to implement the action recognition system is distributed between two processing modules, processing module 0 and 1. Processing module 0 is assigned low level, high bandwidth image capture and communication tasks. This module contains:

- Intel hex upload/download engine: allows data to be uploaded to and downloaded from the FPGA using the extended Intel hex format. This allows the contents of the status and control registers of the various components on the system bus to be read and written to, simplifying testing. This interface can also be used to replace the top level processor module, allowing the user to send configuration and control packets to the FPGA.
- DMA engine: direct memory access controller, can be configured to transfer blocks of data from memory to memory, FIFO to memory, memory to FIFO, or clear a block of memory.
- External synchronous SRAM: 2,048 kB of memory used as a frame buffer storing image data and intermediate results.
- Parallel port: primarily used to transfer image data back to the host PC for SVM training and hardware debugging.
- Camera I2C port: synchronous serial port allowing the FPGA to configure the camera's control registers.
- Camera parallel port: the camera is configured to constantly stream image data to the FPGA. This slave port captures image data, configured with dual 32 kB frame buffers allowing the previous image to be processed whilst the current image is captured. This component also supports Bayer pattern to greyscale conversion and down sampling to lower image resolutions.
- Seven segment display: two seven segment LED displays used to display the motion recognition result and debugging.
- Wishbone to Wishbone bridge: to enable different processing cores to operate in parallel without impacting on system bus bandwidth, functionality can be spread across a number of system buses, i.e. high

bandwidth data accesses are isolated to the local system bus.
- Processing core 0: difference operator, used to generate the MHI.
- Processing core 1: sub-sample operator, used to down sample image data to lower resolutions, using pixel interpolation or selection.

Processing module 1 is assigned high level, lower bandwidth data processing tasks. To maximize system performance access to data is localized to the current system bus, i.e. additional memory is attached to each bus minimizing Wishbone-to-Wishbone bridge transfers. Alternatively, internal dual port BlockRam can also be used to transfers data across systems bus boundaries. This module contains:

- External asynchronous SRAM: edge detection frame buffers
- Internal RAM: intermediate result buffer used in the calculation of the MHI
- Internal ROM: SVM classification data sets, inner product of this data and the MHI is performed to identify motion.
- Processing core 2: inner product, signed 8.8 fixed point multiplication with a signed 24.8 fixed point accumulator.
- Processing core 3: filter, Gaussian or mean smoothing of image data
- Processing core 4: rotate, image orientation about its centre
- Processing core 5: edge detector, Robert's cross or Sobel edge detector operators.

Note that processing cores 3–5 are not used in the motion recognition system, but have been developed as part of the VCL. The processing cores used in processing module 0 and 1 are based on the same hardware architecture as shown in Fig. 7. This greatly simplifies hardware development and testing, allowing the same component to be used for a number of different tasks through firmware modifications. Each processing core contains:

- PicoBlaze processor: depending on the algorithm used (instruction code size), the KCPSM, KCPSM2 or KCPSM3 PicoBlaze processor core can be used.
- Co-processor interface: a generic co-processor interface supporting eight parameter registers, four result registers, control and status registers.
- Look up table ROM: system constants, e.g. sine lookup table, size 16–64 B.
- Scratch pad RAM: intermediate results, size 16–128 B
- Processing core to Wishbone bridge: this bridge has more functionality than the PicoBlaze to Wishbone bridge, supporting automatic read and write pointer updates and offset calculations from these base pointers.

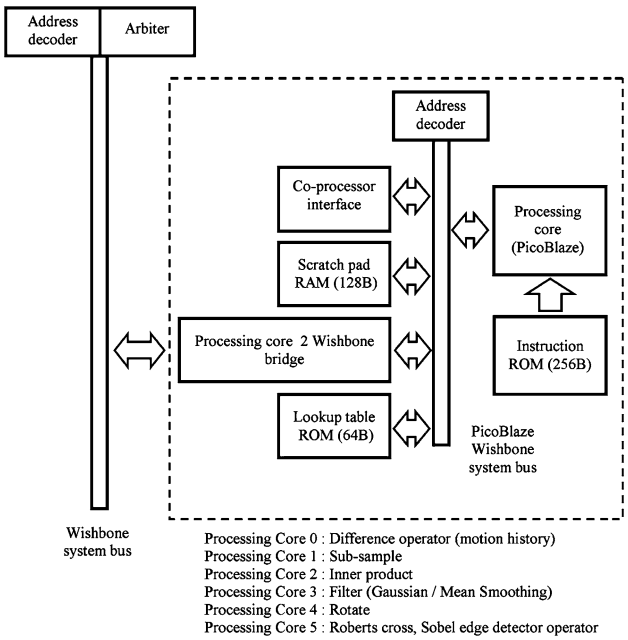

Fig. 7 Processing core block diagram

The Amadeus USE board and action recognition processing pipeline is shown in Fig. 8. A difference operator is performed on the current and previous frames, updating a MHI. The inner product of the MHI and the SVM classification data sets is then performed, the result of each accumulator then has a specific offset applied before a threshold is performed, selecting the stored action that most closely matches the observed motion. In the current implementation this process is operated in a one shot mode, however, this could be easily expanded to include motion detection to start and stop this process, i.e. when the difference between two frames exceeds a threshold the MHI is generated, when it falls below this threshold the inner product and threshold operations are then performed.

The processing performance for each of these system functions is shown in Fig. 9. These results are for a software only implementation, i.e. no co-processor support is incorporated in any of the processing cores. The current hardware implementation uses a 20 MHz system clock and can capture image data at 12 frames/s, i.e. one frame every 80 ms. To allow the system to process data at this frame rate, the inner product calculation performance must be improved. To achieve this level of performance the system can be reconfigured, replicating this processing core improving parallelism. Performance can also be improved through the use of co-processors, moving the signed 8.8 fixed point multiplication operation out of software into dedicated hardware. This hardware was constructed using

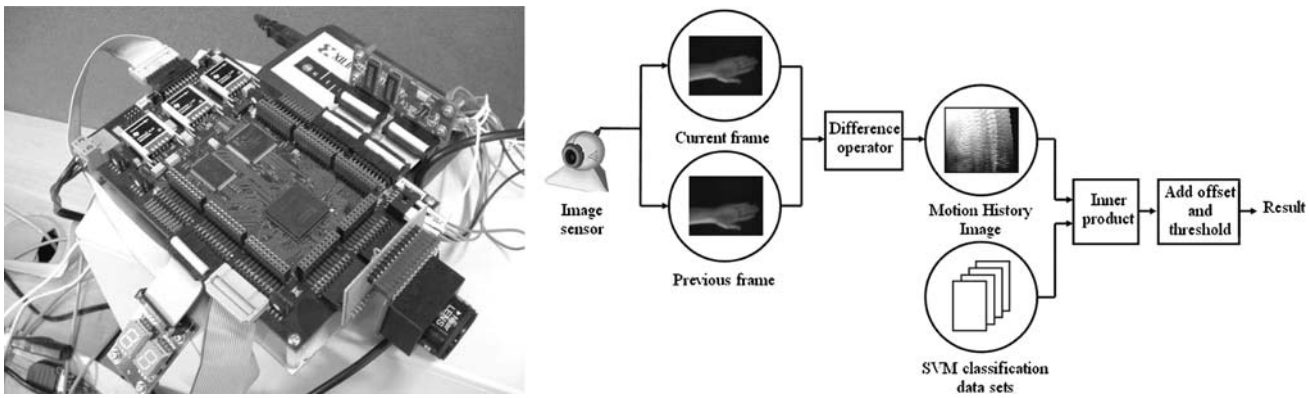

Fig. 8 Amadeus ubiquitous system environment (*USE*) board and motion recognition processing pipeline

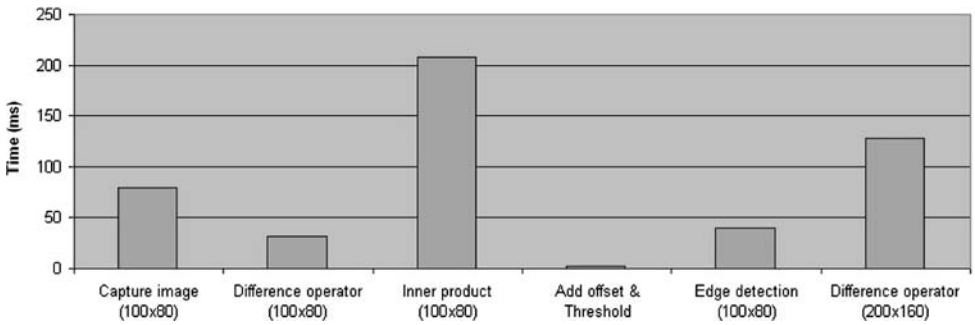

Fig. 9 System function processing performance at 20 MHz

Xilinx's Logicore IP core generator architectural wizard to produce a pipelined multiplier unit. The result of these improvements for an 100 x 80 image can be seen in Figs. 10 and 11. Figure 10 shows the processing performance of a software only implementation performing six inner product calculations using one to six processing cores, i.e. one to six copies of processing core 2 are added to the system bus. Figure 11 shows the processing performance of the same system but with hardware support through the inclusion of a co-processor in processing core 2. Using these techniques the required processing performance can be achieved. The system's main bottleneck is now the camera's interface, limiting image transfers to 12 frames/s. If an improved PCB layout could be supported, additional processing cores would be required to match these new processing requirements. However, these scaling techniques are limited by area and system bus bandwidth restrictions. An alternative solution to increase processing performance would be to "un-roll" the inner product calculation to maximize the available parallelism, i.e. processing multiple vector elements in parallel. Possible co-processor architectures to support these operations are discussed in [12]. It should be noted that the number of processing cores that can be replicated is dependent on the system bus and memory bandwidth since, as more processing cores are added, a point will be reached where

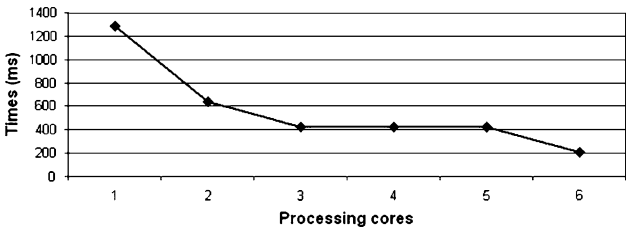

Fig. 10 Software inner product performance

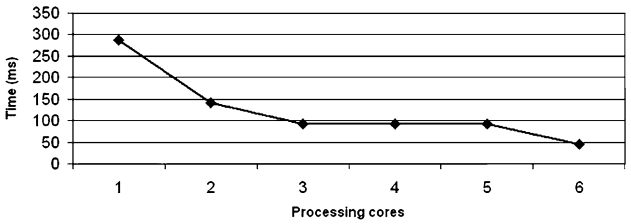

Fig. 11 Co-processor inner product performance

processing on these cores will be stalled until data bandwidth is freed. The performance of these processing cores is also dependent on image size. In the present system, the camera interface captures Bayer pattern images at a resolution of 640 × 480. This is then converted to a greyscale image and down sampled to either 200 × 160 or 100 × 80, as defined via the camera's control registers. The camera input frame rate is therefore constant, however,

the image size can vary from 8 to 32 kB. This increase in data will of course affect the system's processing requirements, e.g. 200 × 160 difference operator's processing requirements increases by a factor of 4. To increase processing performance, the difference operator processing core can be replicated. However, multiple image pairs cannot be processed in parallel due to a data hazard (read after write) on the MHI, i.e. the reader, writer problem. To remove this hazard, the current and previous MHI are divided into segments, with a single processing core allocated to each segment, removing the data hazard problem, as shown in Fig. 12. The processing performance of this system for an image resolution of 200 × 160 with one to six processing cores is shown in Fig. 13. These results highlight the difference between task level parallelism (Figs. 10, 11) and data level parallelism (Fig. 13) when using replicated processing cores. The parallel inner product system uses task level parallelism. In such systems, the number of tasks can be greater than the number of processing cores with tasks being allocated to each processing core as they become available. As a result, processing performance is limited by the maximum number of tasks that can be allocated during each time slice, e.g. with four processing cores and six tasks two time slices will be required, four tasking being processed in the first time slice and two in the second. This granularity is illustrated by the flat regions in Figs. 10 and 11, i.e. for a system with three to five processing cores. In these systems two time slices will be required, resulting in some processing cores remaining idle during these periods. The parallel difference operator system uses data level parallelism. In such systems the number of tasks is equal to the number of processing cores, i.e. the number of segments is matched to the number of processing cores available. As a result, processing performance shows an incremental improvement with the number of processing cores added to the system.

This ability to scale a design to match an application allows the designer to trade off speed and area requirements. In addition to this, each hardware component can also be optimized for speed, size and power considerations, giving a designer greater flexibility in distributing the required processing capabilities amongst the selected processing cores.

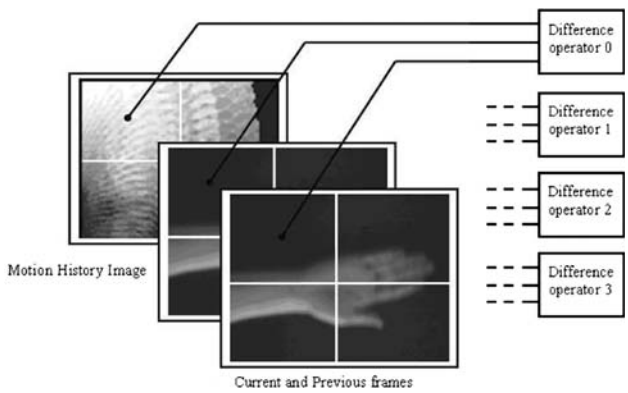

Fig. 12 Allocation of replicated difference operators

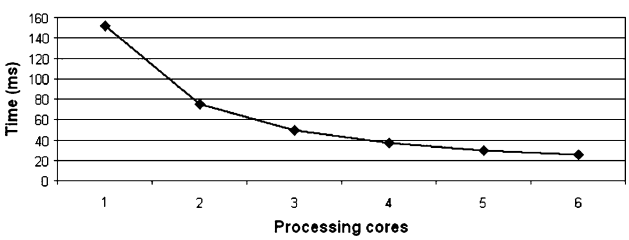

Fig. 13 Software difference operator performance

4.2 Performance testing of the stand-alone system

In order to test the performance of the FPGA implementation of our human action recognition system, we recorded a hand motion dataset. In this dataset, there are only three type of hand motions: horizontal motion, vertical motion and "other motion". We also recognize a "no-motion" case as an extra class.

For each class, we recorded 20 video samples, with the frame size set to 100 x 80 pixels. We recorded the video clips with a variety of backgrounds to test the system robustness to this variability. Figure 14 shows some samples in this dataset.

One of the simplest multiclass classification schemes built on top of real-valued binary classifiers is to train M different binary classifiers, each one trained to distinguish the examples in a single class from the examples in all remaining classes. When it is desired to classify a new example, the M classifiers are run, and the classifier which outputs the largest (most positive) value is chosen. This scheme is referred to as the "one-versus-all" rule.

In our experiment, 15 samples were randomly chosen from each class for training and the other five were used for testing. We repeated the experiments ten times. We carried out the training on a PC using SVM^{light} (the default values were used for all the parameters in this software). Firstly, we extracted MHI features from each video clip. Then we trained three binary linear SVM classifiers based on these features to give a three parameter matrix containing the weight vector $\mathbf{w}$ and bias b. These parameters were stored in the internal memory of the FPGA chip and were used for gesture classification. During the classification, three values were obtained from each SVM classifier and the one with the largest (most positive) value is used to label the motion.

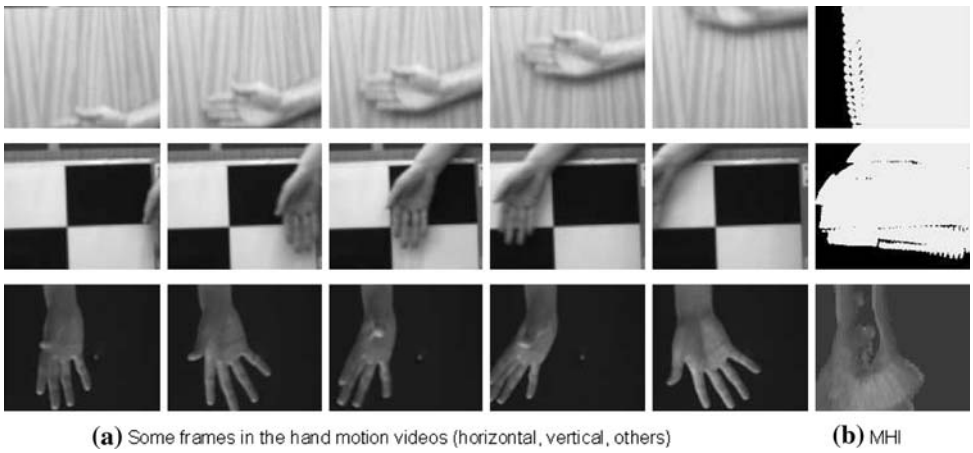

Fig. 14 Some samples in the hand motion dataset and their MHI features

Table 3 Hand motion recognition average confusion matrix

	Horizontal	Vertical	Others
Horizontal	**94**	2	4
Vertical	18	**70**	12
Others	4	18	**76**

Table 4 Hardware performance

	Video capture (s)	Output latency (s)
Sequential	5	2.514
Pipelined	5	0.064

Table 3 shows the average classification rate. The average rate of correct classification for all gestures is 80%. We have compared the outputs of the three binary classifier from both software case (on PC) and hardware case (on FPGA). Note that there is very little differences between the values. The reason for the small difference is that we chose to limit the number of bits in the FPGA implementation. However, the differences do not affect the classification results at all and these results are identical on the PC implementation and the embedded implementation.

Using the system architecture shown in Fig. 6, two hardware configurations can be achieved through firmware modifications, as shown in Table 4. Both of these systems operate in a one shot mode, capturing approximately 5 s of video, i.e. 60 frames at 12 frames/s, at a resolution of at 100x80. The first of these is a test configuration, where each stage of the processing pipeline is performed sequentially, i.e. video capture, difference operator, inner product, offset addition and thresholding. At each stage, intermediate results can be downloaded using the Intel Hex upload/download IP-core and compared to results produced by a Matlab implementation running on a PC. As the results in Table 4 show, this implementation may not be suitable for some real time applications, having an output latency of approximately 2.5 s. The final implementation is constructed to form a virtual pipeline, as illustrated in Fig. 8. To allow this pipeline to operate at 12 frames/s, each stage must be performed within 80 ms. From experimental evaluation of the initial test configuration, it was determined that the final system would require four inner product (with co-processor support) and four difference operator processing cores. To further increase processing performance, the difference operator is overlapped with the video capture. This significantly reduces output latency. Due to internal memory limitations, the final system was limited to three-class classification data sets. However, as Fig. 11 shows, if memory resources were available a system with six inner product processing cores could meet processing requirements.

5 Discussion and conclusions

In this paper, we proposed a system for fast human action recognition. It has applications in security systems, man–machine communication, and other cases of AmI. The proposed method does not rely on accurate tracking as most other works do, since most of the tracking algorithms may incur prohibitive computational cost for the system. Our system achieves a competitive performance with other temporal template based methods, which use representations such as the MHI.

The use of standard processor cores to replace hardware accelerator IP-Cores greatly simplifies system design, i.e. both in terms of component development and debugging. When compared to equivalent systems testing times are significantly reduced, through the use of existing pre-tested processor and peripheral IP-cores and software simulators. The disadvantage of a multiple processor based design is the reduction in unit processing performance when compared to an optimized application specific IP-Core. To

compensate for this, processing cores can be replicated to increase system performance. The advantage of this design approach is that the number of new hardware components that need to be designed is greatly reduced, a common processing core can be used for a number of different tasks through firmware changes. These design advantages come at the cost of increased silicon area, however, with the ever increasing resources available to the designer within modern FPGAs this tradeoff is becoming more acceptable when compared to increasing development costs.

Our embedded human action recognition system performed reliably at 12 frames/s and gave an average action classification rate of 80% over three types of hand gesture. Recognition performance may be improved by recording more data for training.

Our recent work [17, 18] has improved the system classification performance by introducing novel motion features and combinations of motion features, although this is currently only implemented in Matlab on a PC. Implementation of these improved approaches on our reconfigurable video processing architecture is the focus of our future work.

Acknowledgments The authors would like to thank DTI and Broadcom Ltd. for the financial support for this research.

References

1. Aggarwal, J.K., Cai, Q.: Human motion analysis: a review. Comput. Vis. Image Underst. **73**(3), 428–440 (1999). doi: http://dx.doi.org/10.1006/cviu.1998.0744
2. Aizerman, A., Braverman, E.M., Rozoner, L.I.: Theoretical foundations of the potential function method in pattern recognition learning. Autom. Remote Control **25**, 821–837 (1964)
3. Amadeus.: Use—ubiquitous system explorer (fpga development platform). http://www.cs.york.ac.uk/amadeus/projects/centre-use/ (2004)
4. ARC.: Products and solutions: arc configurable cpu/dsp cores. http://www.arc.com/configurablecores/ (2007)
5. ARM.: Processor overview. http://www.arm.com/products/CPUs/ (2007)
6. Blank, M., Gorelick, L., Shechtman, E., Irani, M., Basri, R.: Actions as space-time shapes. In: ICCV, pp. 1395–1402 (2005)
7. Bobick, A.F., Davis, J.W.: The recognition of human movement using temporal templates. IEEE Trans. Pattern Anal. Mach. Intell. **23**(3), 257–267 (2001)
8. Bradski, G.R., Davis, J.W.: Motion segmentation and pose recognition with motion history gradients. Mach. Vis. Appl. **13**(3), 174–184 (2002)
9. Dalal, N., Triggs, B., Schmid, C.: Human detection using oriented histograms of flow and appearance. In: ECCV, vol. 2, pp. 428–441 (2006)
10. Davis, J.W.: Hierarchical motion history images for recognizing human motion. In: IEEE Workshop on Detection and Recognition of Events in Video, pp. 39–46 (2001)
11. Farnell, B.: Moving bodies, acting selves. Annu. Rev. Anthropol. **28**, 341–373 (1999)
12. Freeman, M.: Evaluating dataflow and pipelined vector processing architectures for FPGA co-processors. In: IEEE 9th Euromicro Conference on Digital System Design, Croatia (2006)
13. Joachims, T.: Making large-scale SVM learning practical. In: Schölkopf, B., Burges, C., Smola, A. (eds.) Advances in Kernel Methods—Support Vector Learning. MIT-Press, USA. http://www.svmlight.joachims.org/, oikonomopoulos (1999)
14. Ke, Y., Sukthankar, R., Hebert, M.: Efficient visual event detection using volumetric features. In: ICCV, Beijing, China, October 15-21, 2005, pp. 166–173 (2005)
15. Kodak.: Kodak kac-9628 image sensor 648(h) x 488(v) color cmos image sensor. http://www.kodak.com/ezpres/business/ccd/global/plugins/acrobat/en/productsummary/CMOS/KAC-9628ProductSummaryv2.0.pdf (2006)
16. Meng, H., Pears, N., Bailey, C.: Recognizing human actions based on motion information and SVM. In: 2nd IET International Conference on Intelligent Environments, IET, Athens, Greece, pp. 239–245 (2006)
17. Meng, H., Pears, N., Bailey, C.: A human action recognition system for embedded computer vision application. In: The 3rd IEEE Workshop on Embeded Computer Vision, Minneapolis, USA (2007a)
18. Meng, H., Pears, N., Bailey, C.: Motion information combination for fast human action recognition. In: 2nd International Conference on Computer Vision Theory and Applications (VISAPP07), Barcelona, Spain (2007b)
19. MIPS (2007) Architectures. http://www.mips.com/products/architectures/
20. Moeslund, T., Hilton, A., Kruger, V.: A survey of advances in vision-based human motion capture and analysis. Comput. Vis. Image Underst. **103**(2–3), 90–126 (2006)
21. Ogata, T., Tan, J.K., Ishikawa, S.: High-speed human motion recognition based on a motion history image and an eigenspace. IEICE Trans. Inf. Syst. **E89**(1), 281–289 (2006)
22. Oikonomopoulos, A., Patras, I., Pantic, M.: Kernel-based recognition of human actions using spatiotemporal salient points. In: Proceedings of CVPR Workshop 06, vol. 3, pp. 151–156 (2006)
23. Pears, N.: Projects: Videoware—video processing architecture. http://www.cs.york.ac.uk/amadeus/videoware/ (2004)
24. Schmidt, A., Laerhoven, K.V.: How to build smart appliances. IEEE Personal Commun. **8**(4), 66–71. http://www.citeseer.ist.psu.edu/schmidt01how.html (2001)
25. Schuldt, C., Laptev, I., Caputo, B.: Recognizing human actions: a local SVM approach. In: ICPR, Cambridge, UK (2004)
26. Silicore.: Wishbone system-on-chip (soc) interconnection architecture for portable ip cores. http://www.opencores.org/projects.cgi/web/wishbone/wbspec_b3.pdf (2002)
27. Tensilica.: Xtensa configurable processors—overview. http://www.tensilica.com/products/xtensa_overview.htm (2007)
28. Weinland, D., Ronfard, R., Boyer, E.: Motion history volumes for free viewpoint action recognition. In: IEEE International Workshop on Modeling People and Human Interaction (PHI'05). http://www.perception.inrialpes.fr/Publications/2005/WRB05 (2005)
29. Wejchert, J.: "The disappearing computer", information document, ist call for proposals, european commission, future and emerging technologies. http://www.disappearing-computer.net/mission.html (2000)
30. Wong, S.F., Cipolla, R.: Real-time adaptive hand motion recognition using a sparse bayesian classifier. In: ICCV-HCI, pp. 170–179 (2005)
31. Wong, S.F., Cipolla, R.: Continuous gesture recognition using a sparse bayesian classifier. In: ICPR, vol. 1, pp. 1084–1087 (2006)
32. Xilinx.: Microblaze processor. http://www.xilinx.com/ipcenter/processor_central/picoblaze/picoblaze_user_resources.htm (2007a)

33. Xilinx.: Microblaze soft processor core. http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=micro_blaze (2007b)
34. Xilinx.: Spartan-3 fpga family complete data sheet. http://www.direct.xilinx.com/bvdocs/publications/ds099.pdf (2007c)

Author Biographies

Hongying Meng received his BSc (1991), MSc (1994) in applied mathematics and PhD (1998) in Electronics Engineering all from Xi'an Jiaotong University, Xi'an, China. Since then, he worked at Tsinghua University in China, Dundee University, Southampton University and York University in UK. His research area includes wavelet transform, image compression, medical signal processing, image categorization, computer vision, machine learning and real-time systems. His current work is focus on machine learning based real-time embedded computer vision system.

Michael J. Freeman received the BEng in Electronics Engineering (1995) and PhD in Electronics (1999) from the University of York. He is currently a research fellow in the Department of Computer Science at the University of York. His research interests include hardware architectures for high speed text and vector processing, system on chip and multiprocessor system on chip design for FPGA based systems.

Nick Pears was awarded both a BSc (1985) in Engineering Science and a PhD in Robotics (1989) by Durham University, UK. He then worked as a post-doctoral researcher in the Robotics Research Group, University of Oxford where he developed novel laser range sensors. In 1994, he was elected a fellow of Girton College, University of Cambridge, where he worked on feature extraction and tracking algorithms for scanning range sensors. In 1998 he joined the Computer Science Department University of York, UK. His current research interests are in Computer Vision and Pattern Recognition and include visual navigation, 3D face recognition, visual human–computer interaction and real-time embedded vision.

Chris Bailey is a lecture in Microelectronics and Computer Systems, in the University of York's Department of Computer Science. Prior to 1999, He obtained a PhD degree in stack based Microprocessor design, at the University of Teesside, where he then worked as a Teaching Research Fellow, and a Senior Lecturer. His research interests include ubiquitous systems, image processing hardware, novel processor design, and program optimisation techniques. He has collaborated and co-published work with leading industry partners such as Infineon Uk Ltd. Most recently he led the AMADEUS research initiative with a DTI funded research centre for hardware development in ubiquitous systems.