Journal of VLSI Signal Processing 47, 33–45, 2007

DOI: 10.1007/s11265-006-0016-7

# Designing a Posture Analysis System with Hardware Implementation

J. G. F. COUTINHO, M. P. T. JUVONEN, J. L. WANG, B. L. LO, W. LUK, O. MENCER AND G. Z. YANG
*Department of Computing, Imperial College London, 180 Queen's Gate, London SW7 2EZ, UK*

*Received: 6 February 2006; Revised: 28 June 2006; Accepted: 13 November 2006*

**Abstract.** Posture analysis is an active research area in computer vision for applications such as home care and security monitoring. This paper describes the design of a system for posture analysis with hardware acceleration, addressing the following four aspects: (a) a design workflow for posture analysis based on radial shape and projection histogram representations; (b) the implementation of different architectures based on a high-level hardware design approach with support for automating transformations to improve parallelism and resource optimisation; (c) accuracy evaluation of the proposed posture analysis system, and (d) performance evaluation for the derived designs. One of the designs, which targets a Xilinx XC2V6000 FPGA at 90.2 MHz, is able to perform posture analysis at a rate of 1,164 frames per second with a frame size of 320 by 240 pixels. It represents 3.5 times speedup over optimised software running on a 2.4 GHz AMD Athlon 64 3700+ computer. The frame rate is well above that of real-time video, which enables the sharing of the FPGA among multiple video sources.

**Keywords:** posture analysis, gait analysis, hardware compilation, FPGA, ubiquitous sensor networks

## 1. Introduction

Computer vision and video processing often involve computationally intensive tasks that need to be applied to data streams in real time, with applications ranging from image-guided surgery, security and surveillance to home-care monitoring.

General-purpose computers can support a wide-variety of tasks, but are often too slow or too power hungry for vision applications. FPGAs (Field-programmable Gate Arrays) provide an attractive alternative: they combine the flexibility of software with a speed approaching that of custom hardware technology. It has been shown that, for selected applications, an FPGA at tens of MHz can run up to 1,000 times faster than a microprocessor with a GHz clock [5, 16], while moving critical software loops into hardware can result in average energy savings of 35 to 70% with an average speedup of 3–7 times, depending on the particular device used [19].

In this paper, we focus on the design and implementation of an FPGA-based architecture for simple posture analysis, which determines if the target is standing up, sitting down or lying on the floor. The goal is to build a smart and independent video-camera system that can monitor the daily activities of home care patients. Instead of using body sensors, we rely on images captured by the video camera to identify multiple visual cues to determine posture. A number of clinical studies have shown that changes in posture and gait can indicate the onset or progress of various diseases, such as early signs of neurological abnormalities linked to dementia [21].

Our method [12] is based on previous work on ubiquitous sensing for managed homecare of the elderly [14] and includes the following four contributions:

1. A design workflow for a posture analysis system, summarising the main processing components (Section 3);

2. System architectures and their implementation targeting a Xilinx XC2V6000 FPGA using a high-level hardware design approach (Section 4);
3. Evaluation of the accuracy of our hardware implementations for three posture estimation algorithms (Section 5.1).
4. Performance evaluation and comparison to software for our hardware implementations (Section 5.2).

The rest of the paper is structured as follows. Section 2 covers the background material and provides an overview of our work. Section 3 describes the posture analysis design workflow, while Section 4 discusses its implementation. Section 5 evaluates the performance and accuracy of our approach, and Section 6 summarises the paper.

## 2. Background

Analysis of human motion has in recent years become one of the most active areas of research in the field of computer vision. It provides the opportunity of using an unobtrusive domestic health monitoring system for home-care patients by detecting changes in posture and gait to determine the onset of an adverse event or worsening of an existing condition.

### *2.1. Technique Overview*

In this paper we consider a posture analysis system based on a frame-by-frame technique [14] which comprises three stages: blob detection, blob metric generation and posture classification.

*Blob detection.* A blob describes the shape of an object against a blank background. A common method to extract blobs is to employ *image differencing* and *thresholding* as shown in Fig. 1c and d respectively. The former compares an image with a reference (background) frame to see which parts of the image have changed. This is a simple process of taking the difference between the image frame and the reference frame for each pixel in turn. To take into account variations in the background, such as changes in lighting conditions, the reference image can be adapted progressively. On the other hand, the thresholding process generates a 1-bit (binary) image where pixels with values above the chosen threshold are defined as part of the blob representation, and those below as the background. Binary images are ideal representation for blobs since they are fast to process and store. Noise and distortion can be removed using Gaussian filters as shown in Fig. 1b.

*Blob metrics.* Once a blob is extracted, we can represent it in a number of ways (Fig. 2). Different representations of the blob shape may reveal different features of the shape of the blob. Examples of blob representations include:

– **Projection histogram representation**. Projection histograms are one-dimensional representations that describe the distribution of pixels of an object along the horizontal and vertical axes. Projection histograms can be generated by projecting the binary image on each of the axes. Fig. 2c and d show the horizontal and vertical projections of the blob in Fig. 2a.
– **Radial shape representation**. Radial shape representation describes the outline of the blob by measuring the distance of the outline from the shape centroid at various angles. The radial shape bears resemblance to the blob contour but since we represent the shape as function of the angle instead of along the contour, it will not be the same. Fig. 2b shows an example of the radial shape of the blob of Fig. 2a, plotted in polar coordinates to show the resemblance of the shape and the object.

*Posture classification.* Finally, posture classification determines the posture type (standing, sitting and lying down) for a particular blob target. One common technique to estimate posture is to match a particular blob metric ($T$) against a set of reference patterns ($T_i$) and find an instance in the database that minimises the similarity distance between the blob metric and the reference pattern, that is:

$$c = \arg_i \min d(T, T_i)$$

where $d$ is the similarity measure for a particular blob representation, $i$ is the instance number of the reference pattern in the posture matching database, and $c$ is the resulting posture type.

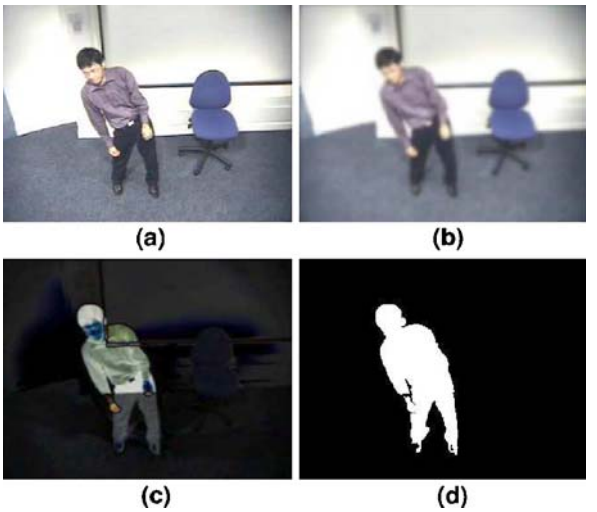



*Figure 1.* An example of blob extraction. An initial Gaussian blur filter is applied on the original image (**a**) to reduce noise and detail in image (**b**). The effect of the image differencing filter is shown in (**c**). Figure (**d**) shows the effects of the thresholding filter, which creates a binary image by dividing the image into two intensities according to the threshold value.

### 2.2. *Previous Work*

The recognition and analysis of human motion and activity are active research topics in the field of computer vision [7, 23]. For example, $W^4$ [10] uses a combination of shape analysis that can be used to track more than one person and recognise various activities. In this case, posture recognition uses projection histograms as well as silhouette shapes.

Pfinder [24] is a sophisticated system that is in use with many applications. It employs a multiclass statistical model of colour and shape to perform tracking of the human body. Pfinder is limited to a single camera and single person setup, although there is a version which uses a stereo camera to obtain three-dimensional models [2]. This single-person tracking assumption is also made by a number of existing tracking systems [14, 17, 18]. Systems to

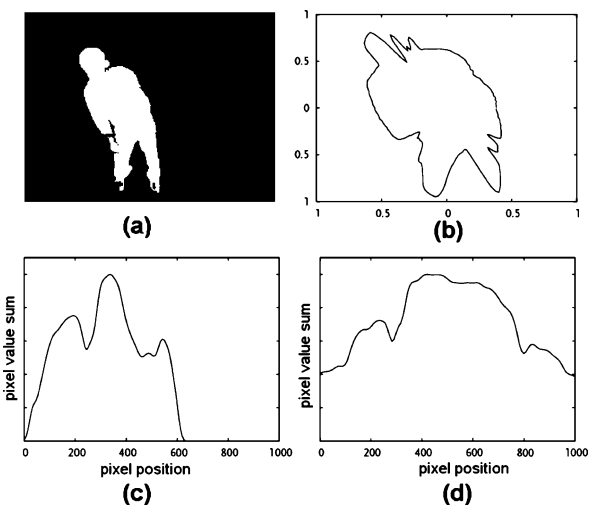



*Figure 2.* Different representations for the blob shown in image (**a**), including a radial shape mapped in a polar coordinate space (**b**), and horizontal (**c**) and vertical (**d**) projection histograms.

track multiple people exist, both for isolated people [3, 13] or people in groups [10].

Some of the more sophisticated systems do not aim for real-time detection. A system containing an automatic calibration scheme and a distributed set of sensors has been proposed which learns common patterns of activity, and can detect patterns that are out of the ordinary [9].

Various hardware implementations based on FPGAs have been proposed. These include systems for applications such as collaborative and reconfigurable object tracking [8], augmented reality [15], and automatic target tracking [22]. However, to the best of our knowledge, the designs proposed in this paper are the first reported FPGA-based implementations for posture analysis.

## 3. Design Workflow

The main workflow is shown in Fig. 3. First, a frame is acquired from a video source. Next a blur filter (*BlurBlock*) is applied to the frame to reduce noise, as explained in the previous section. A difference filter (*DiffBlock*) finds the difference between the blurred frame and a blurred reference image. Second, a threshold filter (*ThreshBlock*) is applied to create a binary image. Third, this image is passed through the *HistBlock* to generate a histogram. Finally, the *RadialBlock* receives the binary image and the histogram to output the blob metrics (both histogram and radial descriptions). Both descriptions are subsequently matched against reference patterns stored in a database (*PostMatchBlock*) to estimate the posture type described in that particular video frame.

Our approach uses three algorithms for estimating posture: vertical projection, horizontal projection, and radial shape. We describe them in Sections 3.3 and 3.4, and their accuracy is evaluated in Section 5.1.

Note that this system assumes a single occupancy environment. However, it can be scaled to deal with multiple occupants by employing an object tracking module, and then process each object individually using our system.

### 3.1. Noise Reduction

The purpose of noise reduction in the posture analysis system (*BlurFilter*) shown in Fig. 3 is to ensure that the binary image representing the blob is as clear as possible (Fig. 1b). In practice an initial low-pass filtering of the image seems to give the best results. Blur filtering usually reduces detail in the image; sophisticated structure-adaptive filters are sometimes used to achieve best results with minimal loss of detail. In the case of the posture analysis workflow, the loss of detail is not a great concern because after thresholding the filtering has minimal effect on the shape of the blob.

### 3.2. Reference Image Updating

Because the image processing system should perform well under a variety of conditions, it should be able to adapt to changing conditions. The actual update rate should be slow enough to adjust to the changing environmental conditions while ensuring that interesting objects that move slowly do not blend into the background. In Fig. 3, the *RefBlock* filter is responsible for updating the reference image.

There are two parameters that can be combined to provide finer-grain control on how the reference image is updated. The first parameter is the update frequency which can be adjusted so that the reference image need not be updated with each frame. The second parameter is the weight used to compute the reference image, which is the weighted

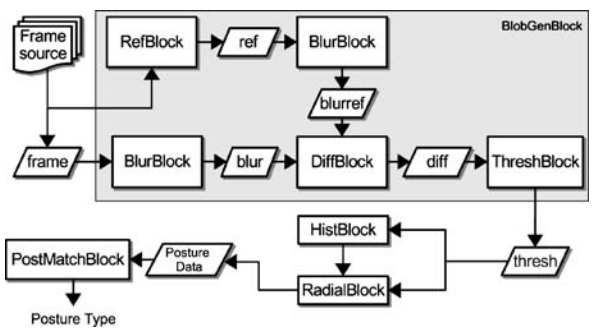



*Figure 3.* The posture analysis system and its image processing blocks.

average between the previous reference image and the latest frame, that is,

$$R(i,j)_{n+1} = wF(i,j)_n + (1-w)R(i,j)_n$$

for reference image $R$, frame $F$ and weight $w$.

### 3.3. Projection Histogram

A projection histogram describes the distribution of pixels across the image. The $n$ th element of the horizontal projection histogram is a count of the number of white pixels in the $n$ th column of the binary (blob) image; similarly, the $m$th element of the vertical projection histogram is a count of the number of white pixels in the $m$th row in the image. In other words, if $T$ is the binary image, then:

$$histogram_H[n] = \sum_{i=0}^{width-1} T(i,n)$$

$$histogram_V[m] = \sum_{j=0}^{height-1} T(m,j)$$

### 3.4. Radial Shape

The radial shape describes the outline of the blob as an array of distances from the centre of the blob over a full rotation around the blob (Fig. 2b).

The first step in calculating the radial distribution is to find the blob centroid coordinates $(c_x, c_y)$. This is done by counting all white (blob) pixels in the image and storing the value in *sum.* Then, we find $c_x$ and $c_y$ so that:

$$\sum_{i=0}^{c_x} histogram_H[i] < sum/2 \leq \sum_{j=0}^{c_x+1} histogram_H[j]$$

$$\sum_{i=0}^{c_y} histogram_V[i] < sum/2 \leq \sum_{j=0}^{c_y+1} histogram_V[j]$$

where $c_x$ and $c_y$ mark the point in the histogram where half of the binary image pixels are on either side, and therefore correspond to the coordinates of the centre of the blob along their respective axis.

Once the centroid $(c_x, c_y)$ is computed, we calculate the radial distribution as follows, where $T$ is the binary image:

1. Build sine and cosine lookup table
2. For each angle $\theta$ in [0..359]:
3. lookup $sin(\theta)$ and $cos(\theta)$
4. $pixel = T(c_x + rcos(\theta), c_y + rsin(\theta))$
5. if foreground *pixel*: increase radius $r$, goto 4
6. else: found shape boundary, repeat for next angle

## 4. Hardware Implementation

In this section we describe the Haydn approach [6] (Section 4.1) and how it is used to derive different design architectures for the posture analysis system (Section 4.2). We also provide details of our development and execution tools (Section 4.3).

### 4.1. Haydn Approach

Hardware synthesis tools tend to fall into two distinct approaches: *cycle-accurate* approach and *behavioural* approach. Each has its own benefits and drawbacks.

The behavioural approach usually employs a software-based language, such as C, to describe hardware functionality, and provides an annotation facility to guide the scheduling process. The behavioural approach provides several advantages, namely: (1) ease of use for software developers, (2) high-productivity for design implementation, and (3) maintainable designs. However, the behavioural approach has a major drawback: hardware synthesis is performed with little human guidance. High-level synthesis often suffers from lack of user control and transparency over the implementation process.

On the other hand, cycle-accurate description languages (such as those based on RTL) give developers more control over low-level implementation details. At this level of abstraction, developers are able to make decisions that would be left to the compiler in a behavioural approach. This allows developers to fine-tune their hardware implementations to achieve an optimal solution. However, cycle-accurate design methodology can have two major disadvantages over high-level synthesis, namely *low productivity and poor maintainability*,

which make it highly ineffective for implementing large designs.

The Haydn approach is unique in that it combines both cycle-accurate and behavioural design methodologies. Developers can opt to use the behavioural approach to rapidly derive a hardware implementation from a high-level design description and constraint annotations. Alternatively, manual intervention can be exerted either at the beginning or at the end of the design cycle to fine-tune a design. We believe that combining both models, manual development and computerised optimisations can be interleaved to achieve the best effect.

We have developed the Haydn-C language [6] to support this methodology. Haydn-C is based on the Handel-C [4] language, but contains significant differences, which we enumerate next. First, Haydn-C is a component-based language like VHDL. This makes it easy for importing and exporting library blocks (such as IP cores) and working with other HDL tools. Second, Haydn-C also provides a meta-language to support source-to-source transformations, additional data structures such as pipelined FIFOs, hardware timers to count the number of cycles for parts of the design, and extended macro capabilities, such as replicators. Most ANSI-C constructs are supported, such as loops, control and assignment statements.

Our hardware design flow is shown in Fig. 4, which performs source-level transformations, cycle-accurate simulation and hardware synthesis for Haydn-C designs. The source-to-source transformation process is guided by annotations in the program that describe design constraints. In particular, the transformation process scans for blocks of code that are enclosed by curly braces and that are annotated with requests for a particular action, such as scheduling. In this case, the block is removed from the rest of the code, analysed and the transformed code is put back in place of the original code. Developers can immediately synthesise the new implementation, simulate or perform another transformation, either by manually revising the code or requesting another computerised optimisation.

The source-level transformation process (Fig. 4) is able to transform both sequential and parallel descriptions by deriving a data-flow graph (DFG) with the *unscheduling* process. A DFG describes the dependencies between operations. The *scheduling* process, on the other hand, places operations in a

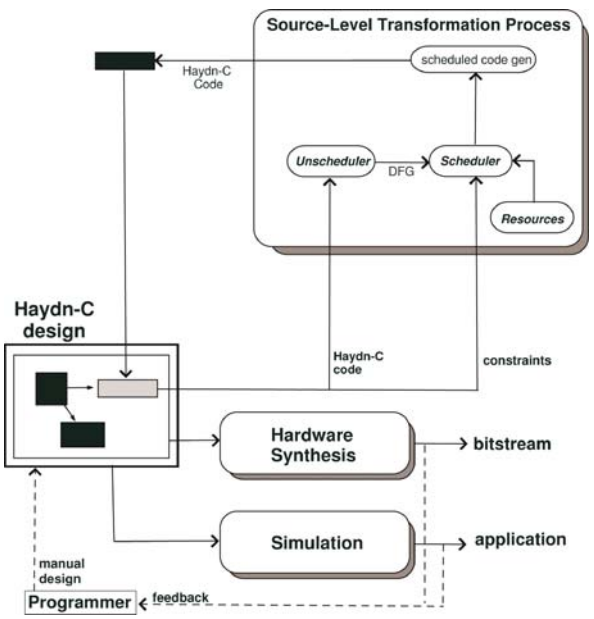



*Figure 4.* This figure illustrates our hardware compilation approach, which performs source-level transformations, hardware synthesis and simulation of Haydn-C designs. Optimisations can be applied to any block of code (enclosed by curly braces) that contains annotations requesting a particular transformation. An example of an user-annotation is shown in Listing 1, line 21. In this case a new block with transformed code is placed over the original code after running the source-level transformation process.

particular time-order without violating program dependencies and user-provided constraints.

### 4.2. Architecture Design

Figure 5 shows the hardware architecture for the proposed posture analysis system. The hardware design is pipelined and contains two coarse-grained stages to maximise throughput. The first stage generates the blob and histogram descriptions, while the second stage computes the radial distribution. Both pipeline stages can work concurrently by storing and accessing two different memory locations alternately. Hence, the design latency is given by the number of pixels in one frame, and subsequently the design is able to output results at a rate of one pixel per cycle.

*Listing 1* Non-optimised (sequential) Haydn-C description of the posture analysis system. Lines 5 and 21 instruct the source-level transformation process to replace each block of code with a pipelined description (Fig. 4) that can generate a result every cycle, that is, with an initiation interval (II) of one.

```
1  set PIXEL_DEPTH = 24;
2
3  task Stage1(idx) {
4     unsigned PIXEL_DEPTH pixel, blurPixel;
5     @scheduler.run(II:1);
6     i = 0;
7     do {
8        ...
9        BlurBlock(pixel, blurPixel);
10       BlurBlock(refPixel, blurRefPixel);
11       DiffBlock(blurPixel,blurRefPixel,diffPixel);
12       ThreshBlock(diffPixel, threshPixel);
13       HistBlock(i, threshPixel);
14       binimage[idx][i] = threshPixel;
15       i++;
16    } while (i != FRAMESIZE);
17 }
18
19 task Stage2(idx) {
20    for (int i = 0; i < 360; i++){
21       @scheduler.run(II:1);
22       int offset;
23          ...
24          offset = ry*FRAME_WIDTH + rx;
25          pixel = binimage[idx][offset];
26       ...
27    }
28 }
29
30 design PostureAnalysis {
31    unsigned 1 memindex = 0;
32    do {
33       par {
34          Stage1(memindex);
35          Stage2(!memindex);
36       }
37       memindex = !memindex;
38    } while (1);
39 }
```

The initial Haydn-C code for the posture analysis design is shown in Listing 1. The code specifies two tasks (lines 3 and 19 respectively) corresponding to each pipeline stage, and the top-level module (line 30). The top-level module instantiates both tasks in lines 34–35. The *memindex* register value indicates which memory to use in the dual-buffer scheme shown in Fig. 5. At each frame, *Stage1* and *Stage2* need to read and write to different buffers. Hence, the *memindex* register value and its negation are passed as input arguments to both tasks respectively. The *par* block in line 33 specifies that tasks *Stage1* and *Stage2* are executed simultaneously. Note that Listing 1 only shows part of the design, and hides details such as variable declarations, definition of the filter blocks, and communication between host and hardware.

To derive different architectures (shown in Table 2) we parameterise the design for different pixel depths by setting the appropriate value in line 1 of Listing 1, and selecting the appropriate multipliers (block or LUT) in the resource table.

The source-level transformation process is guided by user-defined annotations which manage and control the compiler's backend objects, such as the resource table and the scheduler process. Annotations start with an "@" symbol. For instance, the scheduler annotations (lines 5 and 21 of Listing 1) identify parts of the design that need to be optimised. In this case, we specify that we wish to fully pipeline the two blocks of code. Other design configurations can be derived, such as generating a result every $n$ cycles to facilitate resource sharing and multiple video sources. For instance, if the initiation interval is 2, then one can process frames from two sources, where each frame is processed in alternate cycles. This reduces the frame rate by half (assuming one can achieve the same cycle time).

The source level transformation process supports design exploration at three levels: throughput (by setting the initiation interval), controlling resources used by the scheduler and sharing level, and selecting the bitwidth for operation and expressions.

### 4.3. System Development and Execution

We use the Haydn design flow [6] for generating designs that can run in both hardware and software platforms (Fig. 6). The software module that implements the host is built in C++. The hardware, on the

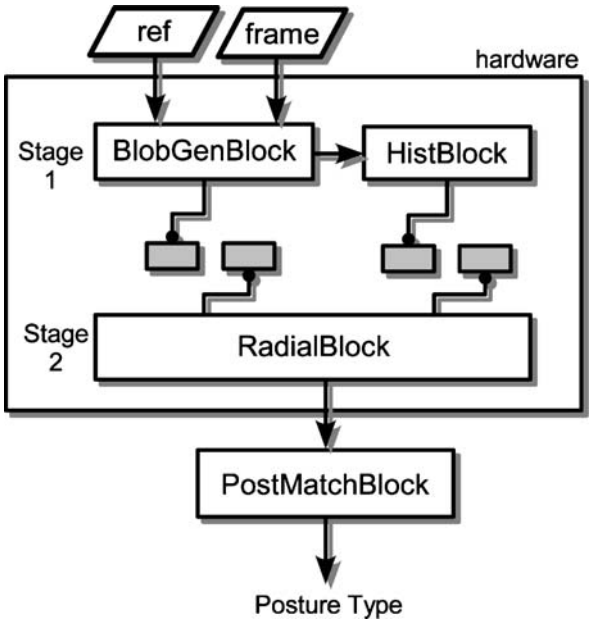



*Figure 5.* The pipelined design for the posture analysis system. The design contains two stages that run concurrently using dual-buffers.

other hand, is described in Haydn-C. A parser converts Haydn-C into Handel-C (HyHC), and we use DK4 capabilities for hardware synthesis and generating a simulation model that runs on a software platform. Simulation involves linking both hardware and host descriptions into a single multi-threaded application to simulate behaviour and communication protocols. On the other hand, hard-

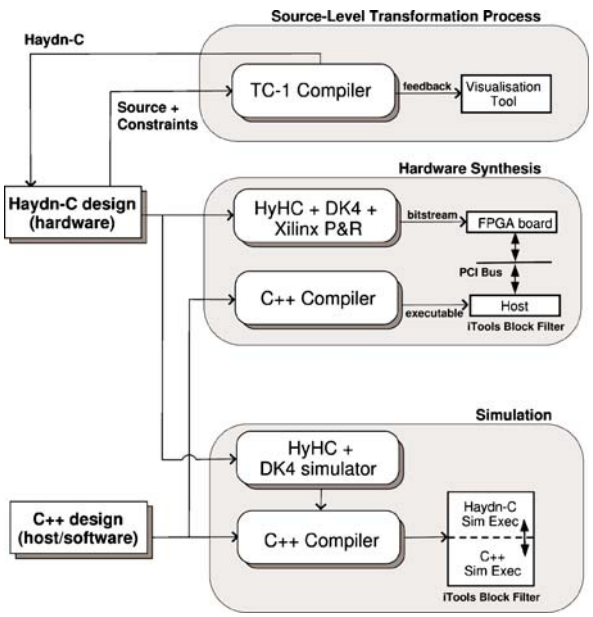



*Figure 6.* This figure shows the Haydn design flow, which performs hardware synthesis, simulation and source-level transformations. The Haydn-C language is used to describe hardware designs, whereas C++ is used to implement the host which runs on a software platform. The hardware synthesis process configures the FPGA device with the posture analysis bitstream and selects the iTools block filter to communicate with the FPGA board. The simulation process, on the other hand, creates an iTools block filter that incorporates the Haydn-C description code for software execution.

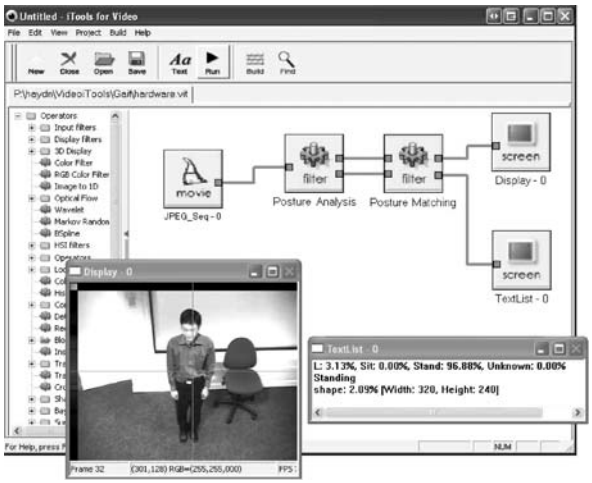



*Figure 7.* The iTools application is used to verify the posture analysis system at software level or using the FPGA board. The right-pane window shows the connections between block filters that implement the posture analysis system. The window showing the video frame depicts the centroid coordinates. The text-box next to the video-frame outputs the estimated posture type (standing).

ware execution involves generating a bitstream to configure the FPGA, which in turn communicates with the host through the PCI bus.

The posture analysis system has been developed using iTools (Fig. 7). This framework has a GUI interface that enables developers to build a system by connecting block filters.

## 5. Evaluation

### *5.1. Accuracy*

The accuracy of our results is an important criterion for building a posture estimation system. When a posture is compared against a set of known postures, the most useful definition of accuracy is the percentage of correctly identified matches.

Image data used in testing the algorithm is a selection of video sequences portraying different postures. The postures are classified as standing, sitting and lying. The test data comprises 2,943 frames totalling 1,830 MB of raw image data. The data are acquired with a stationary Samsung SCC-641 camera and stored in AVI format with Motion JPEG (MJPEG) compression. We choose reference images from the entire data set by selecting typical examples for each posture (standing, sitting and lying). The amount of movement in these images is smaller than that in the test data, so that we can evaluate the tolerance of the algorithm.

Table 1 shows the accuracy of the different algorithms. The horizontal projection description returns the most accurate matches for both the standing and the sitting postures. For the lying posture, radial shape is slightly more accurate than projection. Overall, horizontal projection seems to give a higher accuracy for all three postures. This may partly be a characteristic of the selected set of three postures. The horizontal projection of a standing person is likely to be significantly different from that of a sitting or a lying posture.

*Table 1.* Accuracy of the different posture estimation algorithms.

| | | Accuracy (%) | | |
|---|---|---|---|---|
| Data set (posture) | Frame count | Horiz. project. | Vert. project. | Radial shape |
| Standing | 1,261 | 97.3 | 86.2 | 43.5 |
| Sitting | 841 | 98.6 | 44.9 | 64.3 |
| Lying | 841 | 86.9 | 64.0 | 91.1 |

*Table 2.* Performance comparison of eight FPGA-based designs that implement the posture analysis system.

| Design | Depth (bits/pixel) | Max. Freq. (MHz) | Slices | Frame rate (frames/s) |
|---|---|---|---|---|
| XC2V6-blk1 | 12 | 96.3 | 1,435 | 1,243 |
| XC2V6-blk2 | 24 | 90.2 | 1,543 | 1,164 |
| XC2V6-blk3 | 36 | 80.9 | 1,651 | 1,044 |
| XC2V6-blk4 | 48 | 79.5 | 1,759 | 1,026 |
| XC2V6-lut1 | 12 | 92.6 | 2,857 | 1,195 |
| XC2V6-lut2 | 24 | 89.3 | 2,965 | 1,153 |
| XC2V6-lut3 | 36 | 78.6 | 3,073 | 1,014 |
| XC2V6-lut4 | 48 | 80.2 | 3,181 | 1,035 |

The table includes frame rate and resource utilisation for the Xilinx Virtex-II XC2V6000-4 FPGA. The frame rate is calculated for a 320 by 240 frame for 12, 24, 36 and 48 bits per pixel. Designs with a 'blk' suffix use block multipliers, otherwise LUT multipliers are employed.

The accuracy of the vertical projection, on the other hand, is not very high for the lying posture. It would seem logical to assume that the vertical projection would describe the lying posture in much the same way as the horizontal projection describes the standing posture. One possible reason for the lower accuracy may be the lack of contrast between the person's shirt and the carpet in the image sequence, possibly distorting the blob shape.

The classification process can combine the results of different posture matching algorithms to improve the accuracy of the system by selecting the posture with most support. For instance, if both horizontal and vertical projection algorithms match the *standing* posture then the classification algorithm can be more confident about the outcome. Note that the system is usable even with accuracies below 100%, as we can still derive statistical information about the change of posture over time.

### 5.2. *Performance*

The test data used in this project are captured at a rate of 15 frames per second, a common rate for computer-based applications. Common frame rates vary between 15 and 30 frames per second. The frame size used for this project is 320 by 240 pixels.

Table 2 shows eight designs that implement the posture analysis system described in Section 4. Note that frame rate is projected for the design maximum clock frequency reported by Xilinx tools. The projected frame rate is for data streamed into the processing core, and does not take into account other I/O constraints such as video input and output. We use the Haydn hardware design-flow (Section 4.1) to derive these architectures automatically using two types of multipliers (block and LUT multipliers), and different colour depths (12, 24, 36 and 48 bits per pixel). As expected, different resource and depth configurations provide performance tradeoffs in execution time and area.

We compare the 24-bit (RGB) FPGA version (XC2V6-blk2) against the software versions running on different instruction processors in Table 3, including the Athlon AMD64 3700+ [1], the Intel Xeon [11] and the Trimedia TM1300 [20]. As one can see, the FPGA design outperforms the Athlon 64

*Table 3.* Performance comparison between different platform implementations, including an FPGA and three instruction processors.

| Platform | Clock rate (MHz) | Frame rate (frames/s) | Bandwidth (Mbits/s) |
|---|---|---|---|
| XC2V6-blk2 (FPGA) | 90.2 MHz | 1,164 | 2,046 |
| Athlon 64 3700+ | 2.4 GHz | 330 | 580 |
| Intel Xeon | 2.66 GHz | 264 | 464 |
| Trimedia TM1300 (DSP) | 180 MHz | 30 | 52 |

The frame rate is calculated for a 320 by 240 frame using 24-bit pixels. The bandwidth corresponds to the number of bits processed per second for each implementation.

3700+ by a 3.5 fold speedup, while running at 90.2 MHz clock rate. The software versions have been implemented with full optimisations on all platforms, and frame rate results do not take into account video capturing and rendering.

## 6. Conclusion

This paper describes the design and implementation of an FPGA-based architecture for human posture analysis to monitor and assess the daily activities of home care patients. We show the use of multiple visual cues, such as projection histogram and radial shape description, to estimating changes in posture. The hardware implementation can run 3.5 times faster (1,164 frames/s) than a software version running on 2.4 GHz AMD Athlon 64 3700+ computer (330 frames/s).

Current and future work includes refining our architecture and tools. For example, currently the position of the blob is found by analysing the histograms. A more sophisticated object tracking algorithm capable of tracking multiple people would be an useful extension.

In the future we intend to develop more efficient designs that analyse blob metrics. In particular, blob radial description is a basis for many algorithms, such as skeletonisation, to measure changes in stride and gait frequency. Such algorithms are more useful to track changes in gait compared to simpler posture-based algorithms.

The architecture runs fast enough to analyse more image data than one camera supplies, so another interesting improvement consists of adding support for multiple cameras. Furthermore, smart video-cameras work either independently of each other to monitor a large area, or together to improve detection accuracy. Alternatively, further tradeoffs, involving metrics such as speed, area and power consumption, can be examined in order to target low-cost devices while still keeping performance sufficient for real-time operation.

## Acknowledgements

The support of DTI Next Wave Programme, *Fundação para a Ciência e Tecnologia* (Grant number SFRH/BD/3354/2000), UK Engineering and Physical Sciences Research Council (Grant number EP/C 509625/1 and EP/C 549481/1), Celoxica Limited and Xilinx, Inc. is gratefully acknowledged. Furthermore, we thank the reviewers for their useful suggestions.

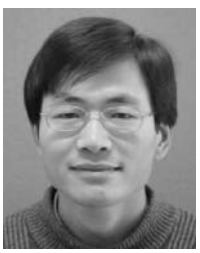
## References

1. Advanced Micro Devices (AMD) Inc., http://www.amd.com.
2. A. Azarbayejani, C. Wren, and A. Pentland, "Real-time 3D Tracking of the Human Body," in *Proc. of IMAGE'COM 96*, 1996.
3. T. Boult, "Frame-rate Multibody Tracking for Surveillance," in *Proc. of DARPA Image Understanding Workshop*, 1998.
4. Celoxica Ltd, http://www.celoxica.com/.
5. C. C. Cheung, W. Luk, and P. Y. K. Cheung, "Reconfigurable Elliptic Curve Cryptosystem on a Chip," in *Proc. Int. Conf. on Design Automation and Test in Europe (DATE)*, vol. 1, 2005, pp. 24–29.
6. J. G. F. Coutinho, J. Jiang, and W. Luk, "Interleaving Behavioural and Cycle-accurate Descriptions for Reconfigurable Hardware Compilation," in *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2005.
7. D. M. Gavrila, "The Visual Analysis of Human Movement: A Survey," *Comput. Vis. Image Underst.*, vol. 73, no. 1, 1999, pp. 82–98.
8. S. Ghiasi, H. J. Moon, A. Nahapetian, and M. Sarrafzadeh, "Collaborative and Reconfigurable Object Tracking," *J. Supercomput.*, vol. 30, 2004, pp. 213–238.
9. E. Grimson and C. Stauffer, "Adaptive Background Mixture Models for Real Time Tracking," in *Proc. of the Computer Vision and Pattern Recognition Conference*, 1999.
10. I. Haritaoglu, D. Harwood, and L. S. Davis, "$W^4$: Real-time Surveillance of People and their Activities," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, 2000, pp. 809–830.
11. Intel Corporation, http://www.intel.com.
12. M. P. T. Juvonen, J. G. F. Coutinho, J. L. Wang, B. L. Lo, W. Luk, O. Mencer, and G. Z. Yang, "Custom Hardware Architectures for Posture Analysis," in *IEEE International Conference on Field Prog. Tech.*, 2005.
13. A. Lipton, H. Fujiyoshi, and H. Patil, "Moving Target Detection and Classification from Real-time Video," in *Proc. of the IEEE Workshop Application of Computer Vision*, 1998.
14. B. Lo, J. L. Wang, and G. Z. Yang, "From Imaging Networks to Behavior Profiling: Ubiquitous Sensing for Managed Homecare of the Elderly," in *Adjunct Proc. of the 3rd International Conference on Pervasive Computing*, May 2005.
15. W. Luk, T. K. Lee, J. R. Rice, P. Y. K. Cheung, and N. Shirazi, "Reconfigurable Computing for Augmented Reality," in *Proc. of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 1999, pp. 136–145.
16. O. Mencer and W. Luk, "Parameterized High Throughput Function Evaluation for FPGAs," *J. VLSI Signal Process.*, vol. 36, no. 1, 2004, pp. 17–25.
17. T. Olson and F. Brill, "Moving Object Detection and Event Recognition Algorithms for Smart Cameras," in *Proc. of DARPA Image Understanding Workshop*, 1997, pp. 159–175.

18. J. M. Rehg, M. Loughlin, and K. Waters, "Vision for a Smart Kiosk," in *IEEE Conference Computer Vision and Pattern Recognition*, 1997.
19. G. Stitt, F. Vahid, and S. Nematbakhsh, "Energy Savings and Speedups from Partitioning Critical Software Loops to Hardware in Embedded Systems," in *ACM Trans. on Embedded Computing Systems*, vol. 3, no. 1, 2004, pp. 218–232.
20. TriMedia TM1300, http://www.tm1300.com/.
21. J. Verghese et al., "Abnormality of Gait as Predictor of Non-Alzheimer's Dementia," *N. Engl. J. Med.*, vol. 347, no. 22, 2002, pp. 1761–1768.
22. J. Villasenor, B. Schoner, K. Chia, and C. Zapata, "Configurable Computing Solutions for Automatic Target Recognition," in *Proc. IEEE Symposium on FPGAs for Custom Computing Machines*, 1996, pp. 70–79.
23. L. Wang, W. Hu, and T. Tan, "Recent Developments in Human Motion Analysis," *Pattern Recogn.*, vol. 36, no. 3, 2003, pp. 585–601.
24. C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland, "Pfinder: Real-time Tracking of the Human Body," in *Pfinder: Real-time Tracking of the Human Body*, vol. 19, no. 7, 1997, pp. 780–785.

**Jose Gabriel de F. Coutinho** received his B.Eng. degree in Computing Engineering from Instituto Superior Tecnico in Lisbon (Portugal) in 1997 and his M.Sc. degree from Imperial College London in 2000. He is currently finishing his Ph.D. studies and working as a research assistant at the Custom Computing group at Imperial College London. His research interests include high-level compilation techniques and tools for reconfigurable and embedded systems, and application domains that can benefit from parallelisation, such as graphics and image processing.

**Matti Juvonen** received his M.Eng. degree in information systems engineering in 2005 and his M.Sc. degree in advanced computing in 2006, both from Imperial College London, UK. He is currently working as a research assistant at the Custom Computing group of the Department of Computing, Imperial College London. His research interests include high-level hardware design methodologies, image processing and computer vision applications.



**Dr. Liang Wang** obtained his Ph.D. from National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences in 2004 and his B.E. and M.E. from the Department of Electronics Engineering and Information Sciences, Anhui University, China in 1997 and 2000, respectively. He worked as a research assistant in the Department of Computing, Imperial College London from July 2004 to September 2005. From October 2005, he has been in the Department of Electrical and Computer Systems Engineering, Monash University, Australia. His main research interest includes computer vision, pattern recognition, image/video processing, machine learning, etc.

**Benny Ping Lai Lo** received his B.Sc. degree in electrical engineering from the University of British Columbia, Vancouver, BC, Canada in 1995 and his M.Sc. degree with distinction in electronic research from King's College London, University of London, London, UK in 2000. He is currently pursuing his Ph.D. degree at Warwick University, Coventry, UK. He worked as a project engineer with Cybermation System Inc., Canada, for 2 years, and later joined the Mass Transit Railway Corporation, Hong Kong SAR, where he was a design officer of infrastructure design—operating control system until 1999. Later, he was a research associate at King's College London until 2001 and a senior researcher at Kingston University, Kingston, UK until 2003 on two EU-funded projects: ADVISOR and PRISMATICA. He is currently working as a research associate in Imperial College London on a DTI-funded project, UbiMon.

**Oskar Mencer (M'96)** received his B.S. degree in computer engineering from The Technion, Israel in 1994 and his M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA in 1997 and 2000, respectively. He founded MAXELER Technologies, in 2003, after 3 years as a member of the technical staff in the Computing Sciences Research Center at Bell Labs. He is a member of the academic staff in the Department of Computing, Imperial College London, U.K., and with the Custom Computing group. His research interests include computer architecture, computer arithmetic, very large scale integration (VLSI) microarchitecture, VLSI computer-aided design (CAD), and reconfigurable (custom) computing. More specifically, he is interested in exploring application-specific representation of computation at the algorithm level, the architecture level, and the arithmetic level.

**Wayne Luk (S'85—M'89)** received his M.A., M.Sc., and Ph.D. degrees in engineering and computing science from the University of Oxford, Oxford, U.K in 1984, 1985, and 1989, respectively. He is a professor of computer engineering in the Department of Computing, Imperial College London, U.K. and leads the Custom Computing Group there. His research interests include theory and practice of customizing hardware and software for specific application domains, such as graphics and image processing, multimedia, and communications. Much of his current work involves high-level compilation techniques and tools for parallel computers and embedded systems, particularly those containing reconfigurable devices such as field-programmable gate arrays.

**Guang-Zhong Yang (g.z.yang@imperial.ac.uk)** received his Ph.D. in computer science from Imperial College, London. His research has focused on medical imaging, robotics, and sensing. He is an associate editor for IEEE Transactions on Medical Imaging and has received several major international awards including the I.I. Rabi Award from the International Society for Magnetic Resonance in Medicine. He is director of medical imaging at the Institute of Biomedical Engineering, Imperial College; chair of the Imperial College Imaging Sciences Centre; and founding director of the Royal Society/Wolfson Medical Image Computing Laboratory at Imperial College. In 2001, he was honored with the Royal Society Research Merit Award chair in medical image computing.