# Custom Hardware Architectures for Posture Analysis

M. P. T. Juvonen, J. G. F. Coutinho, J. L. Wang, B. L. Lo, W. Luk, O. Mencer and G. Z. Yang
*Department of Computing, Imperial College London*
*{mpj01,jgfc,lwang,benlo,wl,oskar,gzy}@doc.ic.ac.uk*

## Abstract

*This paper describes the design and implementation of hardware architectures for posture analysis. Posture analysis is an active research area in computer vision. It can be used in providing health-care solutions, such as monitoring home care patients. We report four contributions in this paper: (a) requirements for a posture analysis system with hardware support; (b) a workflow for posture analysis that fulfils these requirements; (c) various architectures and their implementation based on a high-level hardware design approach; (d) performance evaluation for our derived designs. For instance our best design, which targets a Xilinx XC2V6000 FPGA at 90.2MHz, is able to perform posture analysis at a rate of 1164 frames per second with frame size of 320 × 240 pixels, or 220 frames per second for DVD quality of 720 × 576 pixels per frame. This represents a 145-fold speedup over a software version running on a 3Ghz Pentium-4 computer.*

## 1. Introduction

Computer vision and video processing often involve computationally intensive tasks that need to be applied to data streams in real time. They have many exciting applications, such as vision-guided surgery and robotic navigation; they are also useful in security, surveillance and monitoring systems.

General-purpose computers can support a wide-variety of tasks, but are often too slow or too power hungry for vision applications. FPGAs (Field-progammable 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 1000 times faster than a microprocessor with a GHz clock[3], while moving critical software loops into hardware can result in average energy savings of 35% to 70% with an average speedup of 3 to 7 times, depending on the particular device used[12].

In this paper we focus on the design and implementation of an FPGA-based architecture for human posture analysis. Our overall goal is to build a pervasive visual sensing system that can monitor and assess the daily activities of home care patients. Instead of using body sensors, we rely on images captured by video cameras to identify multiple visual cues, using techniques such as projection histogram and radial shape description, to estimate changes in posture and gait. A number of clinical studies have shown that changes in posture and gait can hold information about the onset or progress of various diseases, such as early signs of neurological abnormalities linked to several types of non-Alzheimer's dementias.

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

1. requirements for a posture analysis system with hardware support (Section 3);
2. a workflow for posture analysis that fulfils these requirements (Section 4);
3. various architectures and their implementation targeting a Xilinx XC2V6000 FPGA using a high-level hardware design approach (Section 5);
4. performance evaluation for our derived designs (Section 6).

The rest of this paper is structured as follows. Section 2 covers background material, and provides an overview of our work. Section 3 explores the requirements of a posture analysis system with hardware support. Section 4 describes a design workflow for producing a hardware system that would meet the requirements, while Section 5 discusses its implementation. Section 6 evaluates the performance and accuracy of our approach, and Section 7 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. As we mention in the previous section, we are interested in providing an unobtrusive domestic health monitoring system for home-care patients using robust computer vision techniques. This includes detecting changes in posture and gait to determine the onset of an adverse event or worsening of an existing condition.

In this paper we consider a posture analysis system based on a frame-by-frame technique [14] comprising three stages: human blob detection, posture type matching based on blob metrics, and behaviour profiling. We explain each stage below.

**Human 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* shown in Fig. 1 (c) and (d) respectively. The former compares an image with a reference (background) frame to see what parts of the image have changed. This is a simple process of taking the difference between two images for each pixel in turn ($O(i,j) = |I(i,j) - R(i,j)|$). To take into account changes in the background, such as 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 part of the blob representation, and below the threshold define 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 can be seen in Fig. 1 (b).

**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. These metrics can then be further used to analyse the shape in various ways. 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. 2 (c) and (d) show the horizontal and vertical projections of the blob in Fig. 2 (a).
- **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, as the shape is described as a function of the angle instead of along the contour, it will not be the same. Fig. 2 (b) shows an example of the radial shape of the blob of Fig. 2 (a), plotted in polar coordinates to show the resemblance of the shape and the object.

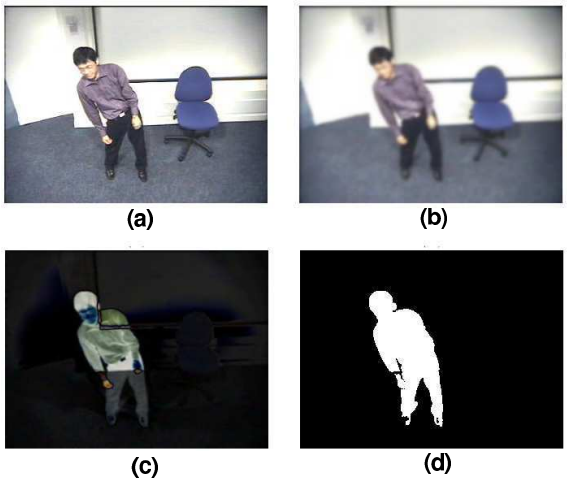


**Figure 1. An example of blob extraction. An initial Gaussian blur filter is applied on the original image (a) to reduce noise and detail in image (b). The effect of the image differencing filter is shown in (c). In (d) we see the effects of the thresholding filter, which creates a binary image by dividing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

**Posture type matching.** Posture estimation can be determined by fusing two or more blob metrics, such as the projection histogram, radial shape and elliptical fitting. In particular, blob metrics are used to classify a given posture ($T$) to one of three main postures (standing, sitting, lying down) by minimising the similarity between it and all reference patterns ($T_i$):

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

where $d$ is the similarity measure computed for the three different representations.

**Behaviour profiling.** To analyse posture changes, an accumulate score based on appearance information is kept so that if a score is beyond a certain range considered as normal, then the system registers it as a change in behaviour. Also, the optical-flow technique can be used to monitor subtle changes in posture from certain parts of the body, such as head, hand and legs.

For the scope of this paper, we focus on human blob detection for hardware implementation, and posture type matching for software execution.

**Previous work.** The recognition and analysis of human motion and activity are currently some of the most active research topics in the field of computer vision [15].

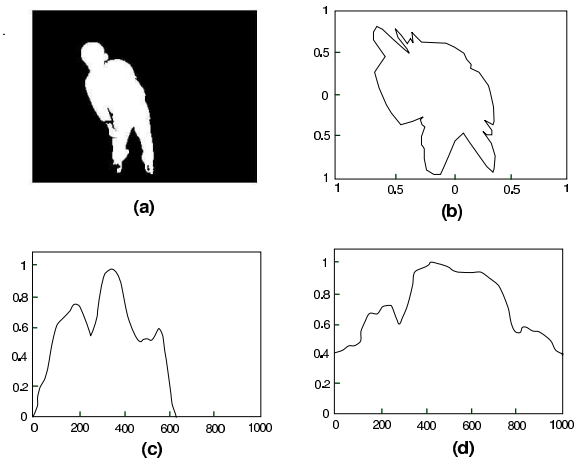


**Figure 2. Different representations for the blob shown in image (a), including a radial shape mapped in a polar coordinate space (b), and horizontal (c) and vertical (d) projection histograms.**

For example, W$^4$ [6] uses a combination of shape analysis and tracking. It is a sophisticated feature-based system that can be used to track more than one person and recognise various activities. Its posture recognition uses projection histograms as well as silhouette shape.

Some of the most sophisticated systems are not aimed for real-time detection. A system containing an automatically calibrating, distributed set of sensors has been proposed that learns common patterns of activity, and can detect patterns that are out of the ordinary [5]. Other systems use statistical modelling with multiple cameras and can detect interesting activity even with random movement in the background.

Numerous other projects exist, making use of a wide range of analysis techniques. Some are limited to a single person in the image, while others are able to analyse people in groups as well. Some systems only consider single isolated frames, while others follow posture changes over several frames [15].

## 3. Design Approach and Requirements

This section introduces the selection criteria and requirements for implementing the posture analysis system into hardware. The basic workflow and scope of the system is shown in Fig. 3.

For an algorithm to be worth considering for hardware implementation, it must be able to behave reliably and predictably in a variety of conditions. Real-life scenes include complications such as image noise, changing illumination and camera shake. The aim of the hardware architecture is to input a video frame, locate an object in the frame, then output data describing the shape of this object. Some of the considerations taken in account when designing the architecture include the following:

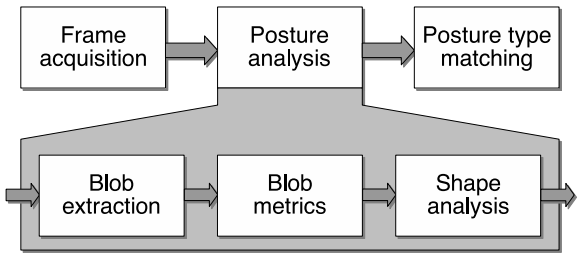


**Figure 3. Our posture estimation system. The posture analysis system is implemented in hardware, while the posture type matching is performed in software.**

**Noise reduction.** The purpose of noise reduction in the posture analysis system is to ensure that the binary image with the blob to be analysed is as clear as possible (Fig. 1(b)). In practice an initial blur 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 as, because of later thresholding, the filtering has minimal effect on the shape of the blob. Applying erosion and dilation filters further corrupts the outline of the blob, and does not remove large groups of pixels. A large kernel causes more dramatic changes to the blob outline, while a small kernel does not remove larger groups of noise pixels.

**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. Reference image updating should therefore be used to enable the system to adapt to gradual changing conditions. The actual update rate should be slow enough to adapt to the changing environmental conditions while ensuring that interesting objects that merely move slowly do not blend into the background.

It is possible to adjust the rate at which the reference image is updated by controlling two variables. First, the update frequency can be adjusted so that the reference image need not be updated with each frame. Second, because the reference image is a weighted average of the previous reference image and the latest frame, the ratio between the two source can be adjusted. A combination of these two parameters provides finer-grain control on how the reference image is updated.

**Colour space considerations.** The selection of the colour space is important to the functionality of the thresholding filter. Ideally, the image should be in a format where the contrast between the object and the background is as high as possible in all circumstances, while noise blends well into the background.

For physiological reasons, the eye is more sensitive to differences in brightness (luminance) than colour (chrominance). This fact is used in many applications, including video capture and transmission (where the luminance channel holds more information that the chrominance channel), and lossy compression. In the context of this project this means that when pictures are acquired from a video camera or an image sequence, the quality of individual channels in the frame may be low. Hence decomposing the image may lead to poor results even if the channel increases contrast.

Because of these problems, and due to the extra overhead required for colour space conversion, the RGB colour model is chosen for this project. This approach works well in most cases for thresholding, and the quality of the results can be improved by thresholding each channel separately and ORing the result.

**Monitoring home care patients.** Since the intended use of this system is to monitor home care patients, we need to take into account two considerations:

- **Privacy.** One of the possible advantages of a posture analysis system, as opposed to a traditional video monitoring system, is that it provides a level of privacy. It should not be possible to reconstitute an image frame from the posture descriptions.
- **Bandwidth.** Most residential buildings are not equipped with high-speed data links. Any data to be transmitted between a posture analysis node and some central monitoring location should have low bandwidth requirements.

The system assumes a single, isolated person in the image. Such an assumption is made by a number of existing tracking systems [10, 11, 14]. Systems to track multiple people exist, both for isolated people [1, 7] or people in groups [6]. Object tracking has also been implement in hardware [8]. Note that in the context of home care monitoring, the assumption of a single person is a reasonable one, as such a system would be used mainly by people living alone.

## 4. Design Workflow

This section introduces the posture analysis workflow taking in account the requirements set in the previous section, and outlines the main points about the design of some of the algorithms used in the system.

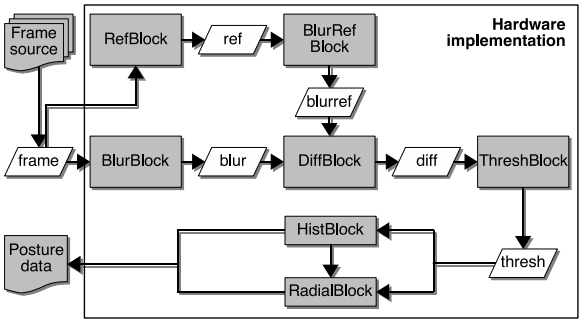


**Figure 4. Our posture analysis system and its image processing blocks, which are implemented in hardware.**

The main workflow is shown in Fig. 3 and from an implementation perspective in Fig. 4. 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. Then, a threshold filter (`ThreshBlock`) is applied to create a binary image. This image is then used by the histogram (`HistBlock`) and radial (`RadialBlock`) algorithms to output posture descriptions.

**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.

**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.

The projection histogram can be used to find the centre of the blob. First, count all white pixels in the image and store the value in `sum`.

Then, using a loop, find $n$ such that:

$$\sum_{i=0}^{n} \texttt{histogram[i]} < \texttt{sum}/2 \leq \sum_{j=0}^{n+1} \texttt{histogram[j]}$$

This is the point in the histogram with half of the binary image pixels on either side, and therefore it marks a coordinate of the centre of the blob along one axis.

To find the radial shape, go over all integral angles from 0 to 359 degrees in a loop. For each angle, follow a straight line from the centre of the blob to the direction of the angle until the edge of the shape (black pixel) is found.

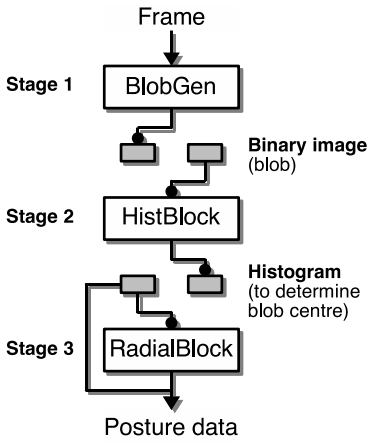


**Figure 5. The pipelined design for the posture analysis system. The design contains three stages that run concurrently using dual-buffers.**

## 5. Hardware Implementation

This section describes the hardware implementation of the posture analysis system, which is shown in Fig. 5. The design is pipelined and contains three stages that are run in parallel using dual-buffering technique. The first stage (`BlobGen`) generates a binary image containing the blob. The second stage produces the horizontal and vertical projection. This data is used not only as a metric to determine posture position, but can also help determine the blob centroid. The last stage determines the radial shape.

Next we highlight some aspects of the implementation that are specific to hardware implementation.

**Ring buffer.** While microprocessors work generally well with algorithms that access data randomly, hardware-based architectures work best with algorithms that are based on processing streams of elements in a pipeline. For the case of a convolution (`BlurBlock`), memory accesses can be reduced by introducing a ring buffer to store pixels needed for subsequent stages of filtering.

The ring buffer is usually implemented using block rams in FPGAs, and must be large enough to store pixels needed for the convolution mask. In the case of a $3 \times 3$ pixel filter kernel and an image of width $w$, the ring buffer must be able to store at least $2w + 3$ pixels. The advantage of this technique is that the 9 memory accesses required to process a pixel (for a $3 \times 3$ kernel mask) are reduced to a single access.

**Reference image updating.** Updating the reference image without using floating-point operations places some limitations on how to perform the operation. If the decay rate is high, for example $w = 1/4$, integer operations can give acceptable results. In this case, the operation is:

$$refP = ((refP \times 3) + (currP))/4$$

Problems arise if $w$ is very low. For example, if $w = 1/128$, then:

$$refP = ((refP \times 127) + (currP))/128$$

In this case the value that the current pixel contributes to the result is so small that rounding errors may mean it does not affect the end result. There are two possible solutions to the problem. The first solution is to set $w$ to a reasonably high value can reduce the effects of rounding errors. This can then be compensated by lowering the reference image update frequency. The second solution is to increase the precision of the stored reference image and treating the reference image pixels as fixed point values. Instead of storing each colour component as an 8-bit value, it could be stored in 12 bits. Now, the operation with $w = 1/128$ is:

$$refP = ((refP \times 127) + (currP << 4))/128$$

This ensures that the pixel values are not stores as integers but rather as fixed-point values with 4-bit decimal portion. In addition, the operation can be done with no multiplies or divisions, with shifts and adds:

$$refP = (((refP << 7) - refP) + (currP << 4)) >> 7$$

When shifts are used, the weight must be a power of two. This is usually an acceptable limitation, though, because this granularity can be compensated with a suitable combination of $w$ and the reference image update frequency.

**Approximating trigonometric functions.** In order to find a radial description of a blob, the algorithm needs to iterate over different angles for a complete rotation. The software version, running on a CPU with a floating-point processor, used sine and cosine functions to convert between Cartesian and polar coordinate systems. Such functions are not readily available for programming hardware.

A number of algorithms exist for approximating trigonometric functions without a floating-point unit [9]. CORDIC [13] is an iterative algorithm for calculating trigonometric functions to an arbitrary precision using only shifts and adds. Table lookup is a simple and effective method for trigonometric approximation. Multipartite tables can be used for greater precision, either using lookup with add or lookup with multiply. Polynomial approximation such as the Taylor series can be used as well.

The radial description algorithm aims to find the distance of the blob outline over a complete rotation. This is done by iterating over a set of angles between 0 and 359 degrees at 1 degree intervals. For the purposes of this algorithm and for the frame sizes considered, this

is a reasonable precision. For each angle, starting from the blob centroid, the algorithm moves outward one step at a time until it finds the blob boundary. The distance of the boundary from the blob centroid is then recorded in an array.

A lookup table of 91 entries is used to do direct table lookup for integral angles between 0 and 359 degrees. This table stores a quarter of a cosine wave between 0 and 90 degrees. All angles between 0 and 359 degrees for both sine and cosine can be generated by shifting and mirroring the quarter-wave along each of the axes.

In order to look up the sine and cosine values in the same clock cycle, dual-ported RAMs are used to store the lookup table. A C program is used to generate a table using any arbitrary fixed-point precision. The current architecture uses a 32-bit lookup table.

**Pipelined radial block.** The radial analysis algorithm is pipelined. The pipeline stages are:

1. Find the quadrant in which the angle lies.

   Resolve the angle to the range [0:90] for all angles. The results are indices to the lookup table and will be used in the next stage of the pipeline. The angles needed are:

   $$\alpha,\ \alpha-90,\ \alpha-180,\ \alpha-270,$$

   $$90-\alpha,\ 180-\alpha,\ 270-\alpha,\ 360-\alpha$$

2. Perform table lookup for both sine and cosine, depending on the quadrant and resolved angles. For example, if the angle $\alpha$ lies in the second quadrant, the values would be

   $$\begin{aligned} \cos\alpha &= -\cos(180-\alpha) \\ &= 0-\text{table}[180-\alpha] \\ \sin\alpha &= \cos(\alpha-90) \\ &= \text{table}[\alpha-90] \end{aligned}$$

3. Multiply the angles by the radius to get the offset from the centre in Cartesian coordinates.

4. Add the offset coordinates to the centroid coordinates to get the absolute position of the point in considerations.

5. Translate each $(x,y)$ coordinate pair into a location in the memory. The memory location is simply $y \times [\text{framewidth}] + x$. For certain known frame widths, this can be made more efficient with a combination of shifts and adds. For example, for a frame width of 320 pixels, the location is $(y << 8) + (y << 6) + x$.

6. Read memory at the location.

7. If the pixel read is black (background colour), we have found a shape boundary so break and repeat for the next angle. Otherwise increase radius and repeat.

**Hardware design with Haydn-C.** We use the Haydn design flow [4] for generating hardware designs. Haydn extends DK3 [2] capabilities of simulation and hardware synthesis with automatic source-to-source transformations. In particular, Haydn supports two main features: deriving architectures that meet performance goals involving metrics such as resource usage and execution time, and inferring design behaviour by generating behavioural code that is easy to verify and modify from scheduled designs such as pipeline architectures. Fig. 6 shows the hardware compilation design flow. All hardware architectures have been implemented using the Haydn-C language [4], which is an extension to the Handel-C language. It contains an annotation language that directs the source-to-source transformation process as well as including powerful macro capabilities. On the other hand, the software module executing on the host, such as the posture type matching, is built using Visual Studio C++. One main feature of the Haydn design flow is that simulation and hardware synthesis can both be performed without changing the hardware (Haydn-C) and software (C++) source-code when targeting the RC2000 board. For hardware synthesis, a bitstream and the host application are generated, and both communicate with each other using the PCI bus. Simulation involves linking both hardware and host designs into a single multi-threaded application to simulate the behaviour and communication protocols.

## 6. Evaluation

In this section we evaluate the performance and accuracy of the posture analysis system.

**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 1,830MB 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 chose 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

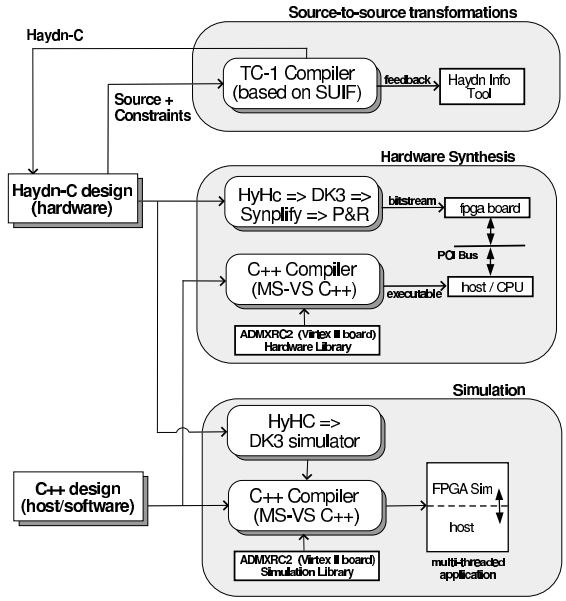


**Figure 6. The Haydn design flow, which performs hardware synthesis, simulation and source-to-source transformations. The Haydn-C language is used to describe hardware designs, whereas C++ is used to implement the host which runs on a CPU.**

the sitting postures. For the lying posture, radial shape is slightly more accurate than projection. 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.

Overall, horizontal projection seems to give a higher accuracy for all three postures. This may partly be characteristic to 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.

The combined posture description is obtained by comparing the three descriptions and selecting the posture with most support from the algorithms. In other words, a posture that matches *lying* in two algorithms and *sitting* in one is considered likely to be the lying posture.

**Performance.** It is important that, regardless of which posture estimation method is used, it should be able to process frames at a rate equal to or faster than the video source to provide real-time computation. The test data used in this project was 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

| Data set (posture) | Frame count | Accuracy (%) Horiz. project. | Vert. project. | Radial shape | Combined description |
|---|---|---|---|---|---|
| Standing | 1261 | 97.3 | 86.2 | 43.5 | **89.2** |
| Sitting | 841 | 98.6 | 44.9 | 64.3 | **85.0** |
| Lying | 841 | 86.9 | 64.0 | 91.1 | **86.6** |

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

is $320 \times 240$ pixels.

Table 2 shows eight designs that implement the posture analysis system shown in Fig. 5, including the projection histogram and radial shape analysis algorithms. Note that frame rate is projected for the design maximum 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 external memory and video input. We exploit the flexibility of FPGA devices to derive different architectures using two types of multipliers (block and LUT multipliers), and different colour depths (12, 24, 36 and 48 bits per pixel).

Note that the projected video processing speed is calculated taking in account the maximum frequency of a design reported by Xilinx tools. However, because of the PCI bus speed restriction, a design targeting an RC2000 board runs slower. For instance, a frame size of $320 \times 240$ is processed at around 850fps when using 24 bits per pixel. On the other hand, the software version is able to process a frame of size $320 \times 240$ at the 8 frames per second on a Pentium-4 3Ghz. As the input video is captured at a rate of 15 frames per second, it is clear that the current software version is not able to perform real-time posture analysis.

In summary, the hardware version running on a RC2000 board executes 106 times faster than the software version, and 145 times faster when considering the maximum frequency reported by Xilinx tools.

## 7. Conclusion

This paper describes the design and implementation of an FPGA-based architecture for human posture analysis that can be used to monitor and assess the daily activities of home care patients. The techniques applied include the use of multiple visual cues, such as projection histogram and radial shape description, to estimating changes in posture. The accuracy of our results is around 86%, and our hardware implementation at about 80–100MHz can run from 106 to 145 times faster than the software version running on a Pentium-4 3Ghz computer.

Current and future work includes refining our ar-

| Design | Max. Freq. (Mhz) | Slices | frames per second (projected) |
|---|---|---|---|
| d12-blk | 96.3 | 1435 | 1243 |
| d24-blk | 90.2 | 1543 | 1164 |
| d36-blk | 80.9 | 1651 | 1044 |
| d48-blk | 79.5 | 1759 | 1026 |
| d12-lut | 92.6 | 2857 | 1195 |
| d24-lut | 89.3 | 2965 | 1153 |
| d36-lut | 78.6 | 3073 | 1014 |
| d48-lut | 80.2 | 3181 | 1035 |

**Table 2. Performance comparison of 8 designs, including resource utilisation for the Xilinx Virtex-II XC2V6000-4 FPGA. The frame rate is calculated for a 320 $\times$ 240 frame for 12, 24, 36 and 48 bits per pixel. Designs with a 'blk' suffix use block multipliers, otherwise LUT multipliers are employed.**

chitecture by replacing some of the simple algorithms with more sophisticated versions. 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 can be used instead.

New posture analysis algorithms can be designed, either to replace the current versions or increase the variety of algorithms. Work is currently under way to design new algorithms to analyse the blob metrics. The radial description of the blob can be used as a basis for many algorithms, including ones that measure changes in stride and gait frequency. Such algorithms may reveal more useful data to track changes in gait compared to simpler, posture-based algorithms.

The architecture runs fast enough to analyse more image data than one camera could supply, so another interesting improvement would be adding support for multiple cameras. These cameras could work either independently of each other to monitor a large area, or together to improve detection accuracy. Alternatively, further speed/area/power consumption tradeoffs can be examined in order to target low-cost devices while still keeping performance sufficient for real-time use.

**Acknowledgements.** The support of DTI NExtwave 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.

## References

[1] T. Boult, "Frame-rate multibody tracking for surveillance", *Proceedings of DARPA Image Understanding Workshop*, 1998.

[2] Celoxica Ltd, `http://www.celoxica.com/`

[3] C.C. Cheung, W. Luk and P.Y.K. Cheung, "Reconfigurable Elliptic Curve Cryptosystem on a Chip", *Proc. Int. Conf. on Design Automation and Test in Europe (DATE)*, 1:24–29, 2005.

[4] J. G. F. Coutinho, J. Jiang and W. Luk, "Interleaving Behavioural and Cycle-Accurate Descriptions for Reconfigurable Hardware Compilation", *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2005.

[5] E. Grimson and C. Stauffer, "Adaptive background mixture models for real time tracking", *Proceedings of the Computer Vision and Pattern Recognition Conference*, 1999.

[6] I. Haritaoglu, D. Harwood and L. S. Davis, "W$^4$: Real-time surveillance of people and their activities", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):809–830, 2000.

[7] A. Lipton, H. Fujiyoshi and H. Patil, "Moving target detection and classification from real-time video", *Proceedings of the IEEE workshop application of computer vision*, 1998.

[8] W. Luk et al, "Reconfigurable computing for augmented reality", *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 136–145, 1999.

[9] O. Mencer and W. Luk, "Parameterized high throughput function evaluation for FPGAs" *The Journal of VLSI Signal Processing*, 36(1):17–25, 2004.

[10] T. Olson and F. Brill, "Moving object detection and event recognition algorithms for smart cameras", *Proceedings of DARPA Image Understanding Workshop*, 159–175, 1997.

[11] J. M. Rehg, M. Loughlin and K. Waters, "Vision for a smart kiosk", *Computer Vision and Pattern Recognition*, 1997.

[12] G. Stitt, F. Vahid and S. Nematbakhsh, "Energy savings and speedups from partitioning critical software loops to hardware in embedded systems", *ACM Trans. on Embedded Computing Systems*, 3(1):218–232,2004.

[13] J. E. Volder, "The cordic trigonometric computing technique", *IRE Transactions on Electronic Computers*, September 1959.

[14] B. Lo, J. L. Wang and G. Z. Yang, "From Imaging Networks to Behavior Profiling: Ubiquitous Sensing for Managed Homecare of the Elderly", *Adjunct Proceedings of the 3rd International Conference on Pervasive Computing*, May 2005.

[15] L. Wang, W. Hu and T. Tan, "Recent developments in human motion analysis", *Pattern Recognition*, 36(3):585–601, 2003.