1st International Conference on Sensing Technology
November 21-23, 2005 Palmerston North, New Zealand

FPGA based Remote Object Tracking for Real-time Control

Christopher T. Johnston, Kim T Gribbon, Donald G. Bailey
Institute of Information Sciences and Technology,
Massey University, Palmerston North, New Zealand
c.t.johnston@massey.ac.nz, k.gribbon@massey.ac.nz , d.g.bailey@massey.ac.nz

Abstract

Most sensing applications require some form of digital signal processing. This processing can be performed on an FPGA rather than a microprocessor or DSP. While real-time tracking and processing is achievable on serial processors, it can be beneficial to take advantage of the parallelism, low cost, and low power consumption offered by FPGAs. The design and implementation of a real-time object tracking algorithm on an FPGA focuses on minimising resource utilisation to allow functionality of the application that uses the tracking information to be added. The effectiveness of this approach is demonstrated through an interactive arcade-style game that allows the user to manipulate the game environment in real time. The game effectively simulates any application that may use the information derived from sensing and tracking for control purposes, and illustrates the effectiveness of an FPGA based solution.

Keywords: FPGA, object tracking, image processing, real-time systems

1 Introduction

A typical sensing application utilises one or more sensors, appropriate hardware and suitable signal processing algorithms to interpret the sensor data and provide meaningful output. Object tracking often involves the use of a camera (a passive sensor in this case), to provide scene data from which the motion of real-world objects is mapped to system controls [1].

Object tracking for control-based applications usually requires the use of a real-time system as sensing delays in the input can cause instability in closed-loop control. This is particularly important if the user must receive sensory feedback from the system. Real-time processing at video rates can be achieved on a serial processor such as a desktop computer. However, as the number of objects that need to be detected and reliably tracked increases, the real-time processing capabilities of even the fastest desktop computer can be challenged.

This is due to several factors such as the large data set represented by a captured image, and the complex operations which may need to be performed on an image. At real-time video rates of 25 frames per second a single operation performed on every pixel of a 768 by 576 colour image (PAL frame) requires 33 million operations per second. This does not take into account the overhead of storing and retrieving pixel values. Tracking algorithms require that several operations be performed on each pixel in the image resulting in a large number of operations per second.

Field programmable gate arrays (FPGAs) provide an alternative to using serial processors. Continual advances in the size and functionality of FPGAs over recent years has resulted in an increasing interest in their use as implementation platforms for real-time video processing [2].

An FPGA consists of a matrix of logic blocks that are connected by a switching network. Both the logic blocks and the switching network are reprogrammable allowing application specific hardware to be constructed, while at the same time maintaining the ability to change the functionality of the system with ease. Performance gains are obtained by bypassing the fetch-decode-execute overhead of serial processors and by using the inherent parallelism of digital hardware to exploit concurrency within the algorithm. As a result FPGAs may achieve the same computation with operating clock frequencies of an order of magnitude lower than high-end serial processors, lowering power consumption.

This paper presents the design and implementation of a real-time FPGA-based remote object tracking algorithm. Design emphasis has been placed on minimising the logic and resource utilisation of the implementation, leaving resources for additional functionality that will use the tracking information in the desired application. A focus on reducing resource utilisation also allows the use of a smaller and correspondingly lower cost FPGA.

To simulate this additional functionality and pragmatically demonstrate the usefulness of the tracking algorithm, we have designed a simple interactive game. The user's arm movements are tracked and used as an input into the game, allowing the user to manipulate the game environment in real-time. The details of this game are outside the scope of the paper but the control aspects directly relating to where the tracking information is used will be briefly discussed in a later section of the paper.

Section two discusses in detail the elements which make up the vision system and tracking algorithm. Section three presents the results of implementing the tracking system. A discussion of the advantages, disadvantages and trade offs of the image processing algorithm and system is given in section four. Finally, section five briefly discusses a number of ways in which the system can be used for interactive control in the game application.

2 Image processing system

Object tracking requires image segmentation to make a distinction between the target objects and the rest of the scene in the captured image. This process partitions the captured image into several disjoint object regions based on common uniform feature characteristics [3]. One simple method is to segment the image based on colour by applying thresholds to each pixel. This is ideal for stream processing as thresholding is a point operation which can be implemented easily on the FPGA.

A block diagram of the complete system is shown below in Figure 1.

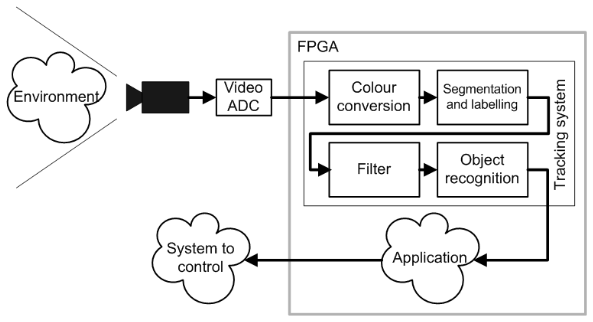

Figure 1: Block diagram of system

The tracking algorithm is broken into four stages: colour conversion, segmentation and region labelling, morphological filtering, and bounding box detection.

The pixel stream from the image capture sub-system is converted to a modified YUV colour space to remove the inherent interdependence between luminance and chrominance in RGB colour space. Segmentation is performed using colour thresholding to associate each pixel with a particular colour class. A morphological filter is then used to remove any noise pixels that are not part of object regions. The filtered stream is then used to construct a bounding box, which encloses each object region so that position, size, orientation and other useful information may be determined for the object.

2.1 Environment

Our aim is to work in an unstructured environment. An unstructured environment (such as no blue/green screen) provides greater system flexibility and portability but can make reliable segmentation more difficult because of the need to distinguish the objects of interest from any other objects that may be present within the image. This limitation may be overcome by restricting the target objects to saturated and distinctive colours to enable them to be distinguished from the unstructured background.

Augmenting the unstructured environment with structured colour in this way is a compromise that enables a much simpler segmentation algorithm to be used. In the context of the game, the user is required to wear or hold fluorescent (highly saturated and intense) colour "markers". Each marker has a different colour to allow the tracking algorithm to differentiate between them. Each object to be identified and tracked must have a unique uniform colour associated with it; this is called its colour class. A discussion of the trade offs associated with this approach appears in section four.

2.2 Image capture

Image capture is performed using a video camera and ADC converter (decoder chip) which digitises the analogue signal from the camera into a stream of 16-bit RGB (5:6:5) pixels. The stream is interlaced with successive fields providing the odd and even lines of the PAL frame.

The algorithm operates on each field of the interlaced frame. This is because each field effectively provides an independent time sample of the environment. The effective image size is therefore reduced to 768 by 288 pixels. Processing each field independently increases the temporal sampling frequency to 50 samples per second. It also avoids "tearing" resulting from the rapid movement of objects of interest between successive fields.

2.3 Colour space transformation

The simplest form of colour segmentation is to independently threshold the red, green and blue components of each pixel. Those pixels that are within all three ranges are classified as belonging to the corresponding colour class. Unfortunately, segmentation results are frequently unreliable when using this approach because any change in the intensity of a colour results in a diagonal movement within RGB space, with a significant effect on all of the components [4]. To allow for this, the colour region has to be quite large with the likelihood of significant background pixels being misclassified into each colour class. This intensity interdependence problem may be overcome by transforming the image to another colour space such as HSI or YUV. Each of these separates the colour components from the intensity or luminance. The transformation from RGB to YUV colour space consists of a co-ordinate rotation to map the RGB cube onto the *Y*, *U* and *V* axes. Transforming from RGB to HSI involves mapping the RGB cube to a cone and is computationally more expensive, although it gives better intensity independence [5].

The YUV colour space is widely used in video and broadcasting [6]. The standard RGB to YUV transformation matrix involves several floating point multiplication operations which are computationally expensive for an FPGA implementation:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad (1)$$

A computationally more efficient alternative is to replace Equation (1) with the modified YUV colour space transform proposed in [4]. This removes the multiplications, replacing them with simple addition, subtraction and shift operations:

$$\begin{bmatrix} Y' \\ U' \\ V' \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 0 & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad (2)$$

The FPGA implementation calculates Y', U' and V' from Equation (2) in parallel for each pixel in the input stream. Both U' and V' are calculated to 6 bits precision. As Equation (2) is a coordinate rotation, any change to the intensity will also affect the values of U' and V'. Therefore, to make the U' and V' less sensitive to illumination, we want to normalise them by the intensity, Y'. By using saturated colours the value of Y' can be lower than that of V'. This can shift the normalised values outside the range -1 to 1. This may be avoided by normalising by

$$Y'' = \max(R, G, B) \qquad (3)$$

instead of Y'. We found 3 bits precision to be sufficient for Y''.

2.4 Colour thresholding

For each colour class, the normalised components are independently thresholded using

$$Y'' > Y''_{\min},\ U'_{\min} < \frac{U'}{Y''} < U'_{\max} \text{ and } V'_{\min} < \frac{V'}{Y''} < V'_{\max} \qquad (4)$$

Only a single threshold is required for Y'' because the target colours are bright. A pixel belongs to a colour class only if it is within all three Y'', U', and V' ranges, and is labelled with a unique ID corresponding to that particular class. As each colour class corresponds to an object being tracked, all pixels within the image are labelled as part of an object region or as part of the background. This reduces the raw pixel data to a unique ID that depends on the number of tracking objects defined. For N colour classes there will be $N+1$ (to include the background label) unique IDs.

Equation (4) includes a division operation which is costly to implement in hardware, using a large amount of FPGA resources and introducing long combinatorial delays if not pipelined [7]. The division can be removed algebraically by multiplying Equation (4) through by Y'' to give

$$Y''U'_{\min} < U' < Y''U'_{\max} \text{ and } Y''V'_{\min} < V' < Y''V'_{\max} \qquad (5)$$

The resulting multiplications are less expensive than the divisions. However, as is shown in the next section, even these multiplications may be eliminated.

2.5 Lookup table optimisation

To perform segmentation and labelling of N colour classes, $4N$ multiplications and $5N$ comparisons must be made on each pixel after performing the colour transformation. For real-time operation, each colour class must have separate hardware because all N sets of comparisons must be performed per clock cycle. These operations can therefore consume significant resources on the FPGA if performed in parallel.

Due to the proliferation of small dedicated on-chip RAM resources on many of today's FPGAs (Xilinx calls this BlockRAM), a lookup table (LUT) can be used to perform the thresholding, normalisation and labelling in a single step. This saves on fabric resource utilisation, combinatorial logic delays and pipeline latency.

The LUT method involves pre-calculating the result of Equation (5) for all valid input values. On initialisation, the resulting values are loaded into local memory on the FPGA (in BlockRAM). All of the tests of Equations (4) and (5) could be performed in one step, requiring 2^{15} entries in the LUT table (3 address bits for Y'' and 6 each for U' and V'). However, since the U' and V' thresholds are independent, they may be separated, reducing lookup to 2 tables of 2^9 entries each. Each entry would produce a single bit result indicating whether or not the colour is within the thresholded range. Multiple colours may be tested simultaneously because several tables in parallel would use the same address (the actual colour of the pixel being classified), but have 1 bit for each colour class being tested. This arrangement is illustrated in Figure 2.

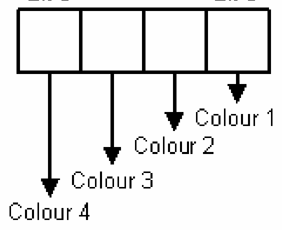

Figure 2: Representation within a LUT element

In our implementation, we test 4 colours simultaneously in this way. A pixel is therefore classified by using the Y'', U', and V' values to simultaneously access the U' and V' tables in the BlockRAM. The two 4-bit results are bit-wise ANDed together. If the result is 0000 the pixel is not part of any of the colour classes and is considered as a background pixel. If there is a single one in any bit position, then the position of the one corresponds to the class of the object. The case of multiple ones is by definition invalid, because a pixel a pixel may belong to at most one class.

The LUT approach to colour classification works with any number of colour classes (subject to resource limitations), and has a constant access time.

2.6 Morphological filtering

Pixels can be labelled incorrectly for several reasons including objects of similar colour to the target in the environment, noise introduced by the image capture sub-system and high contrast edges [8]. To remove these mislabelled pixels we employ a morphological erosion filter with a two by two structuring element window.

This filter will remove any pixel labels that are not part of a group of pixels in a square four element window. This was found to be sufficient to remove most of the noise provided the environment lighting, image capture sub-system (including aperture) and colour thresholds have been defined appropriately (see section four).

The two by two filter is separable which means that filtering can be performed independently using a two element window in each of the horizontal and vertical directions. Thus the current pixel is first compared with the previous pixel (its left neighbour), by using a bitwise AND of the colour class masks. If the previous pixel belongs to the same class, it will be unchanged, but if the class is different, the result will be 0. The result is stored into the corresponding location in a line buffer. The result is also compared with the class label of the pixel from the previous line, obtained from the line buffer. Dual-ported Block RAM is used for the line buffer allowing a simultaneous write and read. The use of the colour class mask enables all classes to be filtered with simple operations in constant time.

2.7 Bounding box

A bounding box [6] provides a simple method for calculating the position, size and aspect ratio of a labelled region. With a little additional processing, the bounding box can also be employed to give the orientation of a non-symmetric object.

The bounding box is defined by the topmost, bottommost, leftmost and rightmost pixels belonging to a particular colour class. The method works by recording the co-ordinates of the first pixel of the labelled region in a raster scan. This gives the top of the box. Then with each successive line the x co-ordinate of any pixel in the labelled region is compared with the recorded co-ordinates of the current left ($\min$) and right sides ($\max$) of the box. These are updated if the detected pixel is outside the present bounds. Finally the last pixel in the labelled region is used to give the y co-ordinate for the bottom of the box.

The bounding box is quite sensitive to noise, as any stray pixel will cause the box to encompass the noise pixel. This is why the morphological filter is required before computing the bounding box. Each target object and thus colour class has its own bounding box and these are calculated in parallel by multiplexing the bounding box hardware depending on the colour class.

After scanning through the entire field each bounding box will have been successfully constructed. Useful information about the bounding box can be extracted during the vertical blanking period. For example:

- The centre of the bounding box will approximate the centre and thus position of the target object. A simple method for calculating this is to add the top and bottom and left and right co-ordinates, shifting each result one bit to the right by one to divide by two.
- The size is given by the width and the height.
- The aspect ratio is given as the ratio of the width to the height.

These properties can then be used to control any application that can map this information to some useful function (see section five for an example).

In order to determine the orientation of the object the following method can be used whilst calculating the bounding box. For each row within the image, if the leftmost boundary is adjusted an accumulator can be decremented, and if the rightmost boundary is adjusted the accumulator is incremented. For a rod-like object at an angle, one side will be extended more frequently than the other. This is reflected by the sign of the accumulator, and provides an indication whether the object is angled to the left or to the right within the bounding box.

3 Results

The resource utilisation for the tracking algorithm is shown below in Table 1. The target device is a XC2S200, a low cost and small sized FPGA.

The implementation uses approximately 10% of the available logic resources. The logic used to implement the segmentation and region labelling appears high but thresholding itself accounts for only 5 logic LUTs and 4 flip flops with the remaining hardware needed to build hardware to initialise the lookup table from off-chip flash memory.

Table 1: Resource utilisation of target device (Xilinx Spartan-II XC2S200) and estimation of instructions required for DSP implementation

	LUT Rams	Logic LUT	Flip Flops	Block RAM	DSP clock cycles
Video decoder		111	78		4
Colour conversion		25	15		8
Segmentation and region labelling		124	147	1	6
Morphological filter		5	12	1	4
Bounding box	39	76	123		6
Calculating position and aspect ratio		19	10		
Total	39	360	385	2	28
Available	4704		4704	14	

For comparison Table 1 also estimates the number of clock cycles required for implementing the per pixel operations on a DSP. The estimate assumes that there are enough registers for storing variables. In addition at least one extra clock cycle is required for looping. Given a video input data rate of 13.5 MHz, a DSP operating at 400 MHz is required to process each pixel. This is a minimum frequency as any additional processing for applications that make use of the tracking information will increase the clock frequency required. The FPGA implementation in comparison operates at only 27 MHz with 90% area left for control applications.

4 Discussion

A number of assumptions have been made in our design and implementation in order to simplify the algorithm and utilise fewer logic resources.

4.1 Lighting

Detecting and segmenting purely by colour introduces a number of interacting variables that must be correctly adjusted. In addition to aperture adjustment of the camera and tuning of the colour thresholds for each target object, lighting within the environment also plays an important role. Although normalisation accounts for moderate changes in light intensity, the colour of the lighting and placement within the environment has a dramatic affect on the reliability of the algorithm. Thus fluorescent lighting is preferred over incandescent lighting because the former does not introduce as much of a red colour shift.

Positioning of the lighting is also important. Lights positioned within the camera field of view can introduce noise pixels into the image. Frontal spot lighting can cause the pixels of the target objects to saturate so that they appear white. Thus diffuse lighting is preferred.

For tuning purposes a $U' V'$ histogram may be helpful in making more objective definitions of the colour thresholds as the distribution of the target objects can be immediately identified.

4.2 Normalisation

As detailed in section two, normalisation can help make the system less dependent on the light intensity. This, however, comes at the cost of colour specificity. Normalisation has the tendency to detect all colours with similar hue. Thus, there is a trade off between intensity insensitivity and colour selectivity.

4.3 Noise

The system is sensitive to noise as the unstructured environment assumes that colour classes will only appear in the target objects (the colour markers). Any non-filtered objects of similar hue (and saturation) to the target objects in the captured image will introduce noise pixels and result in the construction of an erroneous bounding box. The incorrect position and size information derived from this will affect any applications relying on this information for control. This emphasises the importance of environmental set up and tuning of the colour space.

One way to reduce the effects of such noise would be to increase the window size of the morphological filter. Alternatively, a more complex region labelling sub-system could be implemented such as one based on connected component labelling. Both of these would come at the cost of increased on-chip memory and logic usage.

4.4 Tracking information

As mentioned above, the horizontal resolution is better than the vertical resolution in the captured image because each field is processed separately. This results in less sensitivity to movement in the vertical direction. There may also be vertical jitter as the fields are offset vertically. The bounding box is also sensitive to noise on the boundary pixels which can also introduce jitter into the detected position. These introduce an uncertainty or noise into the tracking parameters. More accurate information can be derived by smoothing tracking information over a series of fields, at the expense of additional logic resources.

4.5 Cost

Costing for the Spartan-II family is difficult as it has been superseded by the Spartan-3 family. An equivalent Spartan-3 costs US$15 in sub 500 unit volumes, however as we only use 10% of the FPGA a smaller Spartan-3 FPGA could be used with a cost of US$10 [9]. In comparison, a suitable 400 MHz DSP (Texas Instruments TMS320DM640-400) costs US$23 in 1000 unit volumes [10].

5 Applications

To demonstrate the usefulness of the tracking algorithm we have designed a simple interactive arcade game where the user's arm movements are used as an input into the game. We have made use of the tracking information extracted from the bounding box in a number of ways.

- The centre of the bounding box gives the approximate centre of the target object. Thus the positional information can be used to control a cursor in a similar way to a mouse.
- Interaction with the virtual environment is allowed through manipulation of buttons and sliders. To achieve this we simulate "gripping", "pushing", or "pulling" by detecting the twist of the user's hands upon which the colour markers are worn. The aspect ratio of the bounding box provides information on the amount of twist. We use a ratio of 3:4 or 4:3 to detect when such an action occurs. One limitation with this method is that if the target object extends out of the field of view of the camera, the aspect ratio of the bounding box will be inaccurate. This requires such controls to be positioned away from the edges of the image. The aspect ratio of the bounding box could also be used to provide proportional control to a system.
- Using a non-symmetrical object such as a coloured stick allows us to detect the orientation, if the object axis is perpendicular to the camera's direction. As the slope accumulator is noise sensitive this works better for longer objects than for circular or elliptical objects.
- The distance of an object from the camera can be inferred from the size of its bounding box.

6 Summary

Object identification and tracking requires the use of a real-time signal processing system. Although real-time processing is achievable on serial processors, it can be beneficial to take advantage of the parallelism, low cost, and low power consumption offered by FPGAs.

The successful implementation of this non trivial image processing algorithm illustrates that the digital signal processing required for high rate sensing application can be efficiently implemented on FPGA hardware. This design operates at a clock frequency at least fifteen times lower than frequency required by a serial processor to run the same design, lowering power consumption. In making a compact design, it is necessary to balance optimisations to the algorithm with their effect on the target sensing system. All such optimisations rely on assumptions made about the sensing system, and may restrict the operation of the system.

To reduce resource and logic utilisation the tracking algorithm uses several optimisations. The colour transformation has been simplified to remove the multiplications found in the standard YUV transform. The LUT-based segmentation and region labelling combine two costly operations into one step, eliminating the need for large numbers of parallel comparators.

The resulting optimised implementation can fit on a small and low-end FPGA, such as the Xilinx Spartan-II XC2S200, with sufficient resources still available for an application to make use of the derived tracking information. We have demonstrated this by designing a simple interactive arcade game where the user's movement of coloured markers are used as an input into the game.

7 Acknowledgements

The authors would like to acknowledge the Celoxica University Programme for generously providing the DK3 Design Suite.

8 References

[1] L. Baumela and D. Maravall, "Real-time target tracking," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 10, no. 7, pp. 4-7, 1995.

[2] J. Villasenor and B. Hutchings, "The flexibility of configurable computing," *IEEE Signal Processing Magazine*, vol. 15, no. 5, pp. 67-84, 1998.

[3] K. R. Castleman, *Digital Image Processing*, 1 ed. New Jersey: Prentice-Hall, 1996.

[4] G. Sen Gupta and D. G. Bailey, "A new colour-space for efficient and robust segmentation," *Proceedings of Image and Vision conference New Zealand*, Akaroa, N.Z., pp. 315-320, Nov. 2004.

[5] A. Van Dam and J. D. Foley, *Fundamentals of Interactive Computer Graphics*. Reading, Massachusetts: Addison-Wesley, 1982.

[6] J. C. Russ, *The Image Processing Handbook*, Fourth ed. Boca Raton: CRC Press, 2002.

[7] C. T. Johnston, K.T. Gribbon, and D. G. Bailey, "Implementing Image Processing Algorithms on FPGAs," *Proceedings of the Eleventh Electronics New Zealand Conference, ENZCon'04*, Palmerston North, pp. 118-123, November 2004.

[8] K.T. Gribbon, D.G.Bailey, and C. T. Johnston, "Colour edge enhancement," *Proceedings of Image and Vision conference New Zealand*, Akaroa, N.Z., pp. 297-302, Nov. 2004.

[9] Xilinx Inc, "Online Store," http://www.xilinx.com/, visited on 24/08/2005.

[10] Texas Instruments Inc, "Pricing & Availability," http://www.ti.com, visited on 24/08/2005.