FPGA Implementation of Global Vision for Robot Soccer as a Smart Camera

Miguel Contreras, Donald G Bailey and Gourab Sen Gupta

School of Engineering and Advanced Technology
Massey University, Palmerston North, New Zealand
M.Contreras@massey.ac.nz, D.G.Bailey@massey.ac.nz, G.SenGupta@massey.ac.nz

Abstract. An FPGA-based smart camera is being investigated to improve the processing speed and latency of image processing in a robot soccer environment. By moving the processing to a hardware environment, latency is reduced and the frame rate increased by processing the data as it is steamed from the camera. The algorithm used to track and recognise robots consists of a pipeline of separate processing blocks linked together by synchronisation signals. Processing of the robots location and orientation starts while the image is being captured so that all the robot data is available before the image has been fully captured. The latency of the current implementation is 4 rows, with the algorithm fitting onto a small FPGA.

1 Introduction

The goal of this paper is to improve the accuracy of the robot position and orientation data in a robot soccer environment, by increasing the resolution and frame rate as outlined in a previous paper [1]. Because of the rapidly changing nature of robot soccer, information needs to be processed as quickly as possible. The longer it takes to capture and process the information, the more inaccurate that information becomes. Therefore the accuracy can be improved by increasing the frame rate of the camera and reducing the latency of the processing. This can be achieved by implementing the image processing within an FPGA-based smart camera. By processing streamed data instead of captured frames it is possible to reduce the latency. By also increasing the frame rate it is possible to gather more data and thereby improve the ability to control the robots in the fast changing environment.

The idea of using FPGAs as a platform for a smart camera is not a new one. In fact researchers have used them in various ways in their smart camera implementations. Broers *et al.* [2] outlines a method of using an FPGA-based smart camera as the global vision system for a robot soccer team competing in the RoboCup. Even though they demonstrated that it is possible to use a smart camera to implement the image processing and produce positioning data in real time, their system required multiple FPGAs.

Dias *et al.* [3] describes a more modular approach, where interface cards are connected to the FPGA to perform different tasks, such as Firewire communication, memory modules, as well as the image sensor. This approach does use a single FPGA, however it needs multiple daughter boards in order to function correctly. This can

take up a lot of space and add weight, which can be problematic for a mobile application.

This paper describes an implementation of a smart camera to function as the global vision system for a robot soccer team. To accomplish this, a single off the shelf FPGA development board (Terasic DE0 Nano, although a DE0 was used for initial prototyping because it has a VGA output) was used with a low cost image sensor (Terasic D5M CMOS camera module) to recognise and track the position and orientation of the robot players. The FPGA then transmits the robot data to a separate computer for strategy processing and robot control.

2 The Algorithm

The algorithm is an implementation of that proposed in [1]. It is split up into separate blocks, each handling a particular task or filter, shown in Fig. 1. The first part of the algorithm is completed using pipelined processing of streamed data. This allows the image processing to begin as soon as the first pixel is received, removing the need to buffer the frame into memory.

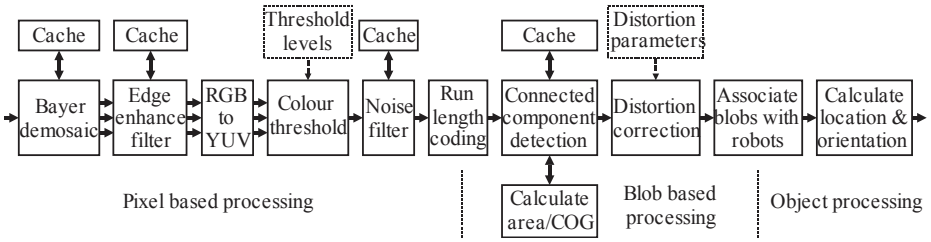

Fig. 1. The block diagram of the algorithm (from [1])

The camera streams 12-bit raw pixel data to the Bayer filter, which derives separate RGB colour channels. The edge enhancement filter removes colour bleeding introduced during the Bayer interpolation process; this allows for more accurate centre of gravity calculation. The signal is then converted into a YUV-like colour space and thresholded to detect the colours associated with the ball and colour patches. A noise filter is used to remove single pixel wide noise and artefacts of the Bayer filter, especially around edges and lines. The filtered signal is then passed through a single pass connected component analysis algorithm which groups the detected pixels together into blobs associated with different colours on top of the robots. The centre of gravity is then extracted from each blob and this is used to recognise and label the separate robots and the ball.

Data flow between blocks in the pipeline is controlled by synchronisation signals. These indicate whether the current streamed pixel is in an active region (valid pixel) or in the horizontal or vertical blanking region (invalid pixel).

2.1 Camera

The D5M digital image sensor can acquire 15 frames per second at its full resolution of 2592×1944. Because it is a CMOS sensor, it allows for windowed readout where a smaller resolution can be captured from the same sensor thus increasing the frame rate. By reducing the resolution to 640×480 it is possible to capture up to 127 frames per second.

Unfortunately the maximum camera height of 2 metres makes the reduced resolution unsuitable because the field of view cannot cover the entire playing area. There are two ways to correct this: The first is by using a wider angle lens, and the second is by implementing another feature of the camera called skipping.

The default lens that comes with the D5M camera has a focal length of 7.12 mm. To see the entire 1.5 m × 1.3 m board, a lens with a focal length of 1.5 mm or lower would be needed. Unfortunately, such a wide angle lens introduces a large amount of barrel distortion. The distortion from a 1.5 mm lens would be difficult to correct so this method on its own is unsuitable.

Skipping is a feature where only every 2nd, 3rd or 4th pixel is read out from the image sensor, effectively producing a smaller resolution from a larger resolution. This makes it possible to output a resolution of 640×480 but sample pixels from a 1280×960 or 2560×1920 area. The disadvantage of using skipping is that it introduces subsampling artefacts onto the image; the greater the skipping factor is, the greater the subsampling artefacts are. This is complicated further with Bayer pattern sensors because of the clustering resulting from the Bayer mask as illustrated in Fig. 2. To completely see the field at 640×480, 4× skipping would be required, however this will also add a lot of area outside of the board and will require a more complex Bayer filter to account for the skipping. A compromise was to use 2× skipping with a 3 mm focal length lens.

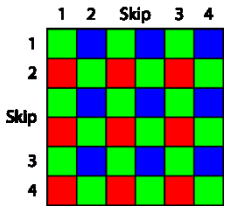

Fig. 2. 2× skipping with a Bayer pattern (from [10])

2.2 Bayer Interpolation

The D5M camera uses a single chip sensor to capture a colour image with a Bayer pattern. Each pixel only captures one of the RGB components as illustrated in Fig. 2, so the missing components must be recovered through interpolation (the difference between the raw image and the interpolated image is shown in Fig. 3).

A bilinear interpolation filter provides reasonable quality interpolation at relatively low cost [6]. For simple Bayer pattern demosaicing, bilinear interpolation simply averages adjacent pixels to fill in the missing values. However because of the skipping introduced by the camera the available pixels are no longer evenly spaced.

An altered bilinear interpolation was created to adjust for the skipping and give evenly spaced output pixels.

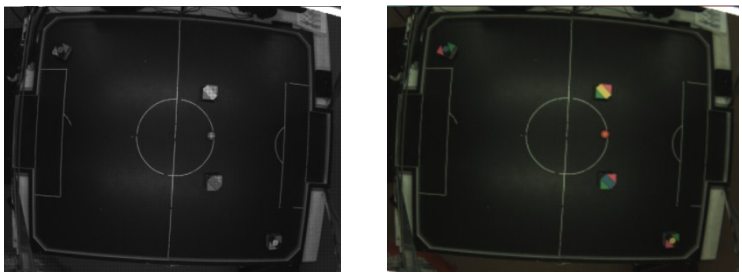

Fig. 3. Left: Raw image captured from the camera, Right: after Bayer interpolation

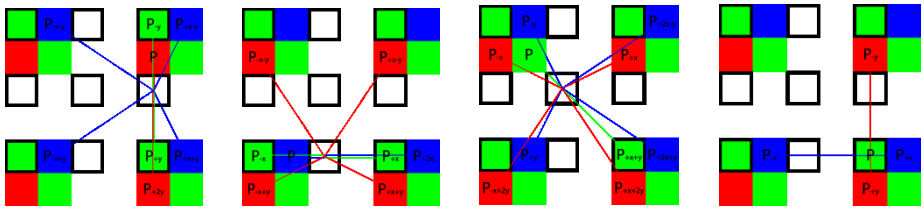

Fig. 4. Pixel locations used to interpolate the RGB values for Green on a blue row (G_B), Blue (B), Red (R), Green on a red row (G_R)

Referring to the pixel locations in Fig. 4, the equations are

$$
\begin{aligned}
G_B : R &= (3P_{+y} + P_{-y})/4 \\
G_B : G &= P \\
G_B : G &= (3P_{+x} + P_{-x})/4 \\
B : R &= (3(P_{+x+y} + P_{-x+y}) + P_{+x-y} + P_{-x-y})/8 \\
B : G &= (P_{+x} + P_{-x})/2 \\
B : B &= (3P + P_{+2x})/4 \\
R : R &= (3P + P_{+2y})/4 \\
R : G &= (P_{+y} + P_{-y})/2 \\
R : B &= (3(P_{+x-y} + P_{+x+y}) + P_{-x-y} + P_{-x+y})/8 \\
G_R : R &= (3(P_{-x} + P_{+x}) + P_{-x+2y} + P_{+x+2y})/8 \\
G_R : G &= (2P + P_{+x+y})/3 \\
G_R : B &= (3(P_{-y} + P_{+y}) + P_{+2x-y} + P_{+2x+y})/8
\end{aligned}
\tag{1}
$$

The altered bilinear interpolation requires a 4×4 window instead of the 3×3 window used by standard bilinear interpolation. This requires 3 row buffers to form the window, and adds a 2 row latency to the algorithm.

To implement the bilinear interpolation as a hardware solution it is important to first optimise the equations so that they can be performed efficiently in real time.

Multiplication by 3/8 and 5/8 can be implemented by an addition and a bit shift. More problematic is the division by 3 required for G_R:G. There are a few ways this can be implemented: The first is to change the divisor to a power of 2, such as multiplication by 5/16 or 85/256. Another method would be to change which green pixels the bilinear interpolation samples from, giving

$$G_R : G = (P_{-x-y} + P_{+x+y}) / 2 \tag{2}$$

However this method makes ¼ of the pixels in the image redundant and therefore ¼ of the image information is effectively lost. Both methods reduce the logic resources and the computational time needed to complete the task, however the second method produced a better quality result, making it the preferred method.

2.3 Colour Edge Enhancement

The colour edge enhancement filter outlined by Gribbon *et al.* [4] was used to reduce the blur introduced by area sampling and bilinear interpolation. The filter utilises a 3×3 window to reduce blurring along both horizontal and vertical edges. The window is comprised or 2 row buffers and adds a single row of latency to the algorithm.

Fig. 5. Left: Blurred image from sampling and bilinear interpolation. Right: After edge enhancement filter.

2.4 YUV Transform

Lighting is very important with all image processing projects. Ideally the light should be spread evenly across the entire area to avoid forming shadows or altering colour values. The RGB colour space is very susceptible to changes in light level. This leads to wide thresholds and results in possible overlap of the threshold values. One solution to this problem is to convert the image into a colour space that is less susceptible to changes in light levels. The YUV colour space maps most of any light level changes onto the Y component leaving the other components for matching the colour. A simplified YUV transform only using powers of 2 is [7], as shown in the equation.

$$\begin{bmatrix} Y' \\ U' \\ V' \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 0 & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3}$$

Because of its simplicity this would only add a single clock cycle delay onto the total algorithm, and allow for simple forward and inverse transformations requiring only additions or subtractions (and shifts).

2.5 Colour Thresholding

The first step is to threshold the image to separate colours from the background. All of the pixels for a particular colour are clustered together in YUV space. Each colour is defined by a rectangular box in YUV, delineated by minimum and maximum thresholds for each component. A colour label is then assigned to each pixel that falls within the threshold limits. This process introduces one clock cycle of latency to the algorithm and can be performed directly on the streamed data.

2.6 Noise filtering

Next the image is filtered to remove isolated pixels from the thresholded image as these can be interpreted as separate blobs. This is performed by implementing a morphological filter to detect single pixel wide noise in either the horizontal or vertical direction. This noise is then cancelled out by changing the colour label to equal the surrounding pixels (see Fig. 6). This allows for most of the noise to be cancelled.

This filter is made up of a 3×3 window which adds another row of latency to the algorithm.

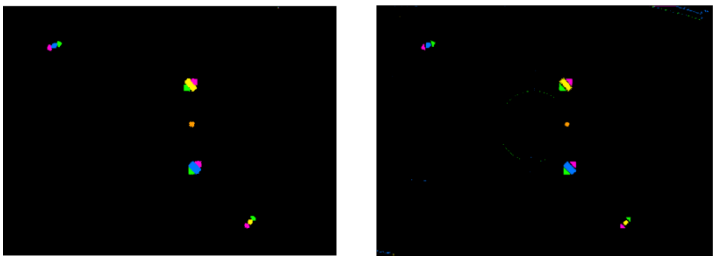

Fig. 6. Left: Labelled image before filtering. Right: After morphological filtering

2.7 Connected Component Labelling

The purpose of the connected component labelling is to group similar adjacent pixels together based on colour. Typically a two pass algorithm is used [9; 8; 5]. The first pass labels pixels into initial groups and the second pass re-labels touching groups with a common label. This allows concave shapes to be labelled as a single group. However since all of the shapes within robot soccer are convex, the second pass is therefore redundant in this application.

A 4 point detection grid can be used to search for adjacent pixels in the streamed data, as shown by Fig. 7(a).

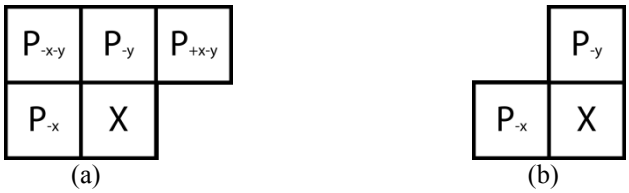

Fig. 7. Left: 4 point detection grid and Right: 2 point detection grid for connected component labelling.

Because the shapes are convex, there is no reason why a group should be linked by a single diagonal connection. Therefore the simpler 2 point detection grid shown in Fig. 7(b) can be used. The previously labelled row is required for the pixel above. To avoid having to re-label a row when a connection with the row above is established, a whole group of pixels is saved into the row buffer at once using run length coding whenever a background pixel is encountered. This reduces the amount of memory used and minimises the number of accesses per frame.

During labelling, the algorithm also accumulates the total number of pixels in the group, and the sum of the X and Y coordinates. When a blob is completed (by detecting that it does not continue onto the next row) this information is used to calculate the centre of gravity of each blob. This processing adds one row of latency to the algorithm for detecting blob completion.

2.8 Blob processing

The final process is to group the blobs into individual robots and calculate their location and orientation. This stage has not yet been implemented however the first step is to calculate the centre of gravity for each blob using data collected during the connected component labelling algorithm (the number of pixels, *N*, within the blob and the sum of the coordinates that each pixel is located at). The equation for calculating the centre of gravity is

$$COG = \left(\frac{\sum x}{N}, \frac{\sum y}{N} \right) \tag{4}$$

A search window approach is used to find blobs with centres of gravity within close proximity. A robot is recognised and labelled once all of the blobs associated with the robot have been recovered.

3 Results

With the camera fully working we are able to see some very promising results. With the current implementation, up to but not including the centre of gravity, we are utilising 3610 LUTs which is only 29% of the total DE0's logic units, and 23% of the FPGA's memory blocks.

This design is capable of operating at 127 frames per second, which is the maximum that this camera is capable of for this resolution. However in order to display the output on a computer screen for debugging purposes it is necessary to limit the frame rate to 85 frames per second, as this is the maximum frame rate the LCD display allows. For the final implementation a display will not be necessary, therefore the frame rate of the camera can be increased to operate at the full 127 frames per second.

In total there are 4 rows of latency added to the algorithm. This means that the data for a robot is available 4 rows after the last pixel for the robot is read out from the camera. Therefore, it is possible to have located and processed all the robots before the frame finishes capturing.

To test the accuracy of the algorithm the blob data was captured from each frame and analysed over a period of 20 seconds. The differences in blob size and position between frames were compared with robots at different areas of the playing field. With the robot in the centre of the playing field (where the light is brightest) the standard deviation for the x and y coordinate is 0.15 mm with a max error of 2.4 mm. A robot was placed in one of the corners furthest away from the centre of the camera and light. The standard deviation for the x and y coordinates in this location was 0.15 mm with a max error of 2.3 mm.

4 Conclusion

In conclusion this paper has described an algorithm that accurately detects the positions of robots in real time, with only 4 rows of latency. Even though at this stage the project is a work in progress it is already possible to operate the camera at 85 frames per second and achieve 2.4 mm accuracy in a worst case scenario. Future study on this project will include automatic setup of the camera and its thresholds as well as implementing a distortion correction calibration to adjust for the added parallax error introduced by the lens.

Acknowledgements

This research has been supported in part by a grant from the Massey University Research Fund (11/0191).

References

[1] Bailey, D., Sen Gupta, G., and Contreras, M.: Intelligent camera for object identification and tracking. In: 1st International Conference on Robot Intelligence Technology and Applications; Gwangju, Korea, Advances in Intelligent Systems and Computing vol. 208, pp 1003-1013 (2012).

[2] Broers, H., Caarls, W., Jonker, P., and Kleihorst, R.: Architecture study for smart cameras. In: Proceedings of the EOS Conference on Industrial Imaging and Machine Vision; Munich, Germany, pp 39-49 (2005).

[3] Dias, F., Berry, F., Serot, J., and Marmoiton, F.: Hardware, design and implementation issues on a FPGA-based smart camera. In: First ACM/IEEE

International Conference on Distributed Smart Cameras (ICDSC '07); Vienna, Austria, pp 20-26 (2007).
[4] Gribbon, K.T., Bailey, D.G., and Johnston, C.T.: Colour edge enhancement. In: Image and Vision Computing New Zealand (IVCNZ'04); Akaroa, NZ, pp 291-296 (2004).
[5] He, L., Chao, Y., Suzuki, K., and Wu, K.: Fast connected-component labeling. Pattern Recognition 42(9): 1977-1987 (2009).
[6] Jean, R.: *Demosaicing with the Bayer pattern*. University of North Carolina: (2010).
[7] Johnston, C.T., Bailey, D.G., and Gribbon, K.T.: Optimisation of a colour segmentation and tracking algorithm for real-time FPGA implementation. In: Image and Vision Computing New Zealand (IVCNZ'05); Dunedin, NZ, pp 422-427 (2005).
[8] Park, J., Looney, C., and Chen, H.: Fast connected component labelling algorithm using a divide and conquer technique. In: 15th International Conference on Computers and their Applications; New Orleans, Louisiana, USA, pp 373-376 (2000).
[9] Rosenfeld, A. and Pfaltz, J.: Sequential operations in digital picture processing. Journal of the Association for Computing Machinery 13(4): 471-494 (1966).
[10] Terasic: TRDB-D5M 5 Mega Pixel Digital Camera Development Kit. Vol. Version 1.2. Terasic Techologies (2010).