Intelligent Camera for Object Identification and Tracking

Donald G Bailey, Gourab Sen Gupta, and Miguel Contreras

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

Abstract. Intelligent cameras extend the concept of smart camera by directly processing the pixels as they stream from the sensor. Working in a synchronous streamed pipeline processing mode, an FPGA incorporated into the camera is able to operate at the camera pixel clock rate. With careful design, this scheme minimizes memory accesses and reduces the latency over DSP based smart cameras. The transformation of the software image processing algorithm to an efficient intelligent camera implementation is demonstrated for global vision within robot soccer. The resulting intelligent camera requires no memory outside of the FPGA, and is able to provide the position and orientation of the objects while the image is being streamed from the sensor.

Keywords: FPGA, robot vision, robot soccer, real-time vision

1 Introduction

Processing speed is critical in many vision applications. This is especially so when vision is used as a sensor in a control system, as is frequently encountered in robotics, machine vision and surveillance. The processing latency limits the usefulness of the data derived by image processing, because increased delays within a feedback loop can severely impact the controllability of a system. The consequences are poorer performance, or in extreme cases, instability.

Mobile robotics imposes additional constraints on vision processing. System size and weight are often limited, and since mobile robots are usually battery powered, power consumption is important. The complexity of many vision algorithms often requires high performance computing to manage the high data rates from cameras. This usually entails processors running with high clock speeds just to maintain the high pixel throughput. High clock speeds result in higher power requirements.

This state of affairs is exacerbated by the increasing resolution of low cost digital cameras. While the increased resolution generally improves the performance of many vision algorithms, this comes at the cost of an increased computational burden. The increased processing cost reduces the time available for other processing tasks, such as strategy and control functions.

To overcome some of these problems, processing is increasingly being moved to within the camera. Consequently, the last decade or so has seen the advent of so-

called “smart cameras”, where the camera is no longer solely responsible for capturing the images, but performing some of the processing, and communication of the processed images or results (Bramberger et al. 2006).

A general purpose processor struggles to keep up with the processing demands of image processing applications. For this reason, digital signal processors (DSPs) are commonly used for implementing smart cameras (Wills 1999; Bramberger et al. 2004; Novak and Mahlknecht 2005; Bramberger et al. 2006). The architecture of DSPs has been optimized for signal processing in a number of ways. The use of a Harvard architecture doubles the memory bandwidth by separating instruction and data memories. The CPU also has single cycle multiply and accumulate functions which speeds up filtering and other related operations.

Low-level vision processing operates independently on individual pixels, enabling the associated parallelism to be readily exploited (Downton and Crookes 1998). This has led to parallel processors which work simultaneously on multiple parts of an image. One example of this is described in (Kleihorst et al. 2001). They developed a dedicated chip that operates a bank of 320 processors in parallel on the pixels within one or more rows using a SIMD (single instruction multiple data) architecture. The input had a serial to parallel conversion to convert the incoming pixel stream to make the image rows available to the processor in parallel. A similar unit transformed the parallel processed data back to a serial stream for output.

In recent years, field programmable gate array (FPGA) technology has matured to the stage where it is practical for image processing tasks. FPGAs have been used in a wide range of ways within smart cameras. One has been through the use of a SIMD architecture to exploit spatial parallelism (Fatemi 2007). For these, a set of identical processing elements (PEs) is used in parallel. Another common architecture for image processing is the window processor (Dias et al. 2007). Here a series of dedicated PEs is used to perform identical processing to the pixels within a window, which are then reduced by another PE to give a single window output. Such an architecture enables a wide range of filters to be implemented, including linear filters, grayscale morphology, and sum of absolute difference. The window architecture exploits functional parallelism (Bailey 2011a), where the same function is applied to all of the pixels within a window. At the other end of the FPGA spectrum, dedicated logic is developed for the particular application, and implemented on the FPGA. One example of this is described in (Leeser et al. 2004). Such approaches generally have the smallest resource requirements, but are less general purpose. However, blocks of logic, such as memory management, camera interfacing, and some basic operations can be reused from one design to the next.

In software, most images are operated on at the image level. That is, an operation is applied to an image by reading the image from memory, performing the operation, and writing the results back to memory. The time required for such processing is dominated by memory accesses, rather than the actual processing performed. With general purpose processors, this also strongly relies on the processor cache to prevent the algorithm speed from being dominated by slow memory. Within a smart camera, though, processing can be moved from an image based computation to a pixel based computation (Bramberger et al. 2004). Here, rather than processing operations on

complete images, a sequence of image processing operations are performed on each pixel, with the intermediate results stored in local registers. The resulting reduction in memory bandwidth required can give significant savings. Where necessary, memory bandwidth can be increased further by operating multiple banks as ping-pong buffers.

Most smart cameras save the incoming image into a frame buffer before beginning processing, even if they begin processing before the complete frame has been captured. On an FPGA, it is possible to process the image as it is directly streamed from the camera. If all of the processing can be implemented in this manner, then the memory requirements (and associated bandwidth required) can be significantly reduced (Bailey 2011a). The strict timing constraint requires not just porting the algorithm from software into the camera, but transforming the algorithm so that it can use stream processing. We term this type of smart camera as an "intelligent camera".

In this paper we will demonstrate this transformation process through the description of an intelligent camera for the global vision processor for robot soccer. The rest of the paper is structured as follows. Section 2 briefly overviews the typical architecture of robot soccer, in particular the robot soccer systems which use global vision. It outlines the changes to the system that result from using an intelligent camera. Section 3 outlines the vision processing algorithms performed by the intelligent camera in this application. The transformation of these algorithms for stream processing is described in some detail. Section 4 describes our implementation of an intelligent camera. Section 5 concludes the paper by discussing the costs and benefits of using an intelligent camera.

This is the first description (that we are aware of) of using an FPGA based intelligent camera for global vision for robot soccer. A further novel aspect of this paper is the description of the algorithm transformations from conventional software to an intelligent camera for this application.

2 Robot soccer architecture

Since robot soccer was introduced to the world in the mid nineties, most teams have relied on using global vision to track the robots and the ball. In a global vision system, a camera is mounted over the centre of the soccer platform at a prescribed height so that the playing field (robots' workspace) is in its field of view. The video signal from the camera is fed to the PC based game controller. For analog cameras, the video signal is digitized using a frame grabber card before further processing. FireWire (IEEE 1394) is by far the most popular digital camera interface used. Where the field size is too big to be seen by one camera, some teams have resorted to using multiple cameras and combining the processed data output from them (Ball et al. 2004).

The typical architecture of a robot soccer global vision system is shown in Fig. 1. The image from the camera is first stored in a frame buffer. The vision processing software then analyses the image to identify objects of interest and this information is passed on to the strategy processor which decides the behaviour of individual robots (Sen Gupta et al. 2005). Based on the roles assigned to the robots, the global control processor generates the commands for the movement of the robots. The commands

are sent to the robots by the communication layer.

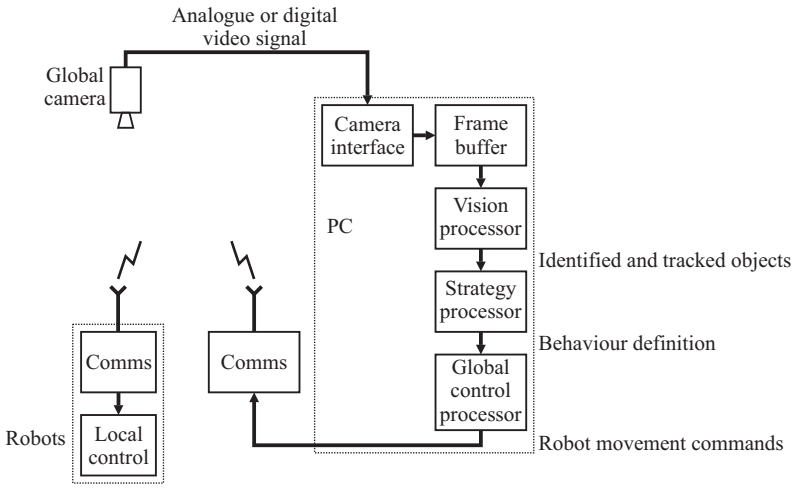

Fig. 1. Typical architecture of a robot soccer global vision system.

The robot soccer game has graduated from 3-a-side in the early years of its inception to 11-a-side. Consequently the field size has also increased several fold. Recently, the trend has been to move to local vision based systems, partly dictated by the revised and challenging rules of the game. In a local vision based system, robots carry the vision system on board the mobile platform to perceive the world around them. This places many constraints on the system hardware and software such as size and processing throughput. A local vision system has been described in (Weiss and Jesse 2004). (Novak and Mahlknecht 2005) have implemented a high-speed onboard vision system comprising a small digital CMOS camera and a very fast and low power signal processing unit.

While there are several examples of use of smart cameras for local vision, research in the area of smart cameras for global vision is still in its infancy. (Wills 1999) designed and partially built a smart camera for robot soccer global vision using a DSP. His smart camera eliminates the frame grabber and PC hardware from the system. (Weiss and Hildebrand 2004) have presented the architecture of a flexible global vision system for robot soccer; the design is such that it can be adapted for multiple cameras and also for local vision systems.

The reduced system from using an intelligent camera is shown in Fig. 2. The intelligent camera incorporates the image sensor and the processing hardware to process the image. The intelligent camera generates the data, notably the coordinates and orientation of the identified objects, which is passed to the strategy layer of the software running on a PC. This effectively removes a lot of the processing overheads from the PC and offloads them onto the intelligent camera. The time available to the PC program for strategy and global control processing is increased resulting in quicker and better control of the robots.

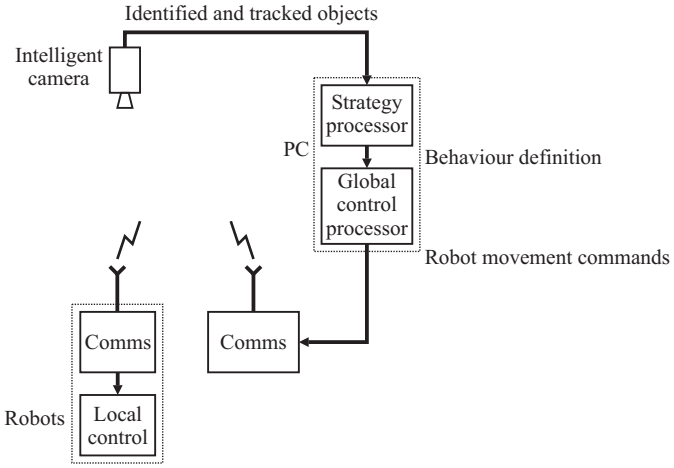

Fig. 2. Reduced system from using an intelligent camera.

With FPGAs making tremendous progress in terms of speed and number of processing blocks, it is feasible now to implement the vision processing in FPGAs and derive the benefits of parallel processing in hardware. In this paper we present the design of a FPGA based intelligent camera for global vision applications and enumerate how software algorithms may be transformed to run on FPGAs utilizing hardware parallelism.

3 Algorithm transformation for intelligent camera

The software algorithm used by the vision processor is outlined in Fig. 3. An RGB colour image from the camera is captured into a frame buffer by a frame grabber card. This image is converted from RGB to a simplified YUV colour space using only integer additions and shifts (Sen Gupta et al. 2004). The RGB colour space is not good for segmentation since changes in lighting significantly affect all 3 components. By converting to YUV, most of the lighting change is reflected in Y with only smaller changes to U and V. This allows significantly better colour selectivity when thresholding to detect the colours. The individual coloured pixels are detected by thresholding the Y, U and V channels independently, effectively using rectangular boxes within YUV space. The detected pixels are then grouped together and assigned unique labels using connected components labeling.

Processing within the next section is on each connected component, or blob, rather than on pixels. First, the coordinates of the centre of gravity of each blob are calculated, with the resulting coordinates corrected for parallax, lens and perspective distortion (Bailey and Sen Gupta 2004). The blobs are then grouped together and associated with robots based on proximity. From the set of blobs, the robot position and orientation can be estimated.

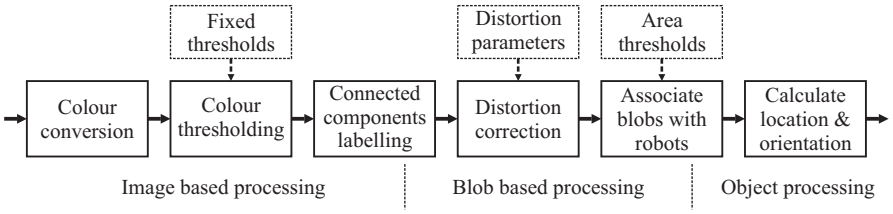

Fig. 3. Software algorithm implemented within the vision processor.

From an image processing perspective, the basic algorithm can be implemented on the FPGA. Since the raw pixel stream directly from the camera is being processed, it is necessary to extend the colour conversion module to include Bayer pattern demosaicing.

However, rather than directly port the software algorithm, the algorithm needs to be transformed to exploit parallelism. Bailey has identified 9 transformation principles to consider when transforming an image processing algorithm from hardware to software implementation (Bailey 2011b). These are

1. Exploit appropriate forms of parallelism

Image processing algorithms consist of a sequence of operations. Building a separate processor for each operation and pipelining the results from one operation accelerated the processing by enabling each processor to operate in parallel. Other forms of parallelism spatial parallelism (unrolling the outer loop which steps through pixels, and build a separate processor for each instance), and functional parallelism (unrolling the inner loop to implement the low level operations in parallel).

The software algorithm in Fig. 3 can be directly be pipelined by building hardware on the FPGA for each operation. Functional parallelism is used to enable each operation to process one pixel per clock cycle.

2. Use stream processing where possible

Stream processing converts the processing from an image basis to a pixel basis. This means operating the pipelines at a pixel level of granularity rather than at the image level. This significantly reduces latency, because it is unnecessary to wait for the whole image to be available before beginning the processing. Stream processing introduces a hard timing constraint, which can be overcome through low-level pipelining. This spreads the time required for each operation over several clock cycles, while maintaining a throughput of one pixel per clock cycle.

Stream processing is the basis of intelligent cameras, because the processing starts as soon as the pixels start arriving from the camera. By operating on the pixels as they arrive from the camera, external memory accesses are minimized.

3. Reduce memory access through local caches

For many image processing operations, the output pixel depends on several input pixels. Filters are a good example; caches save the pixels which are going to be reused later into memory blocks on the FPGA to minimize external memory bandwidth. Custom cache design is made easier by the regular access patterns of most low level image processing operations (simple row buffers are sufficient in many

cases).

The algorithm above requires caches for the filtering associated with demosaicing, and for label propagation as part of connected components labeling. Edge enhancement and noise reduction filters are also introduced into the algorithm to improve the segmentation accuracy. These also require row buffers to cache previous rows.

4. Strip mining and multiplexing

Strip mining is another form of loop unrolling, where separate hardware is built for operating on separate sections of data. When processing pixels, usually each pixel only belongs to one class, enabling a single processor to be multiplexed between the associated data.

This is the case when calculating the area and centre of gravity of each of the blobs. Each pixel only belongs to one blob, so the pixel label can be used to select the corresponding data to be updated.

5. Rearrange algorithm and substitute operations to simplify processing

Many algorithms can be simplified by rearranging the order of the operations, or replacing complex operations with simpler approximations.

In the robot soccer algorithm, this is exploited in several ways. First, rather than calculate the true YUV, which requires multiplications, the simplified YUV can be implemented purely with additions (Johnston et al. 2005). Connected components labeling typically requires two passes through the image. By extracting the data associated with each blob as the image is being labeled, the second pass is no longer necessary (Bailey and Johnston 2007; Ma et al. 2008). In fact the connected components processing can be simplified further since each of the blobs is convex.

6. Reduce data volume through coding

In some applications, processing can be accelerated by compressing the data. In particular, run-length coding has been found useful in a number of applications, including connected components labeling (Appiah et al. 2008).

Run-length coding can simplify the connected components labeling of convex blobs. If a run overlaps matching pixels in the previous line then the whole run can be added to the blob, completely eliminating the need for merger processing.

7. Transform the complete algorithm, not just individual operations

This principle suggests investigating interactions between operations, rather than implementing each operation in isolation (even if using pipelined stream processing).

8. Select data and memory structures based on H/W not S/W requirements

In software almost all data structures are based within a single monolithic memory. On an FPGA, there are a large number of independent small dual-port memories which can be used to implement many of the data structures. The independence of the memory blocks gives a potentially wide bandwidth, although with stream processing it is still necessary to keep in mind the pixel rate – only one memory access may be made per port per clock cycle.

Caching is one example of the use of this principle. Another is using a block of memory to maintain the data structures associated with each connected component.

When implemented correctly, this enables data associated with a region to be output even before the image has been completely scanned (Ma et al. 2008).

9. Use software for software tasks and hardware for hardware tasks.

Not all algorithms map well to hardware. Low level pixel processing is ideally suited to FPGA implementation. The higher level object based processing often consists of complex primarily sequential code, which if mapped to hardware would result in the hardware sitting idle for much of the time. These tasks are best implemented in software.

The robot level processing is one example of this. In this case, the processing to correct for distortion and determine the robot position and orientation is relatively simple. It could either be implemented in hardware, or in software on a soft core processor implemented within the FPGA.

The resulting transformed algorithm is shown in Fig. 4. It is shown in a little more detail than its software counterpart in Fig. 3. Note that the complete algorithm is implemented using pipelined stream processing, and no external frame buffer or other memory is required.

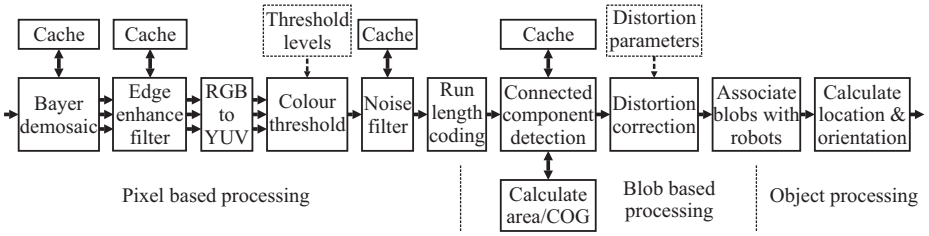

Fig. 4. Hardware implementation of the global vision algorithm on an FPGA.

4 System Implementation

We are currently implementing the algorithm in Handel-C, compiled for the Cyclone IV FPGA on an Altera DE0-Nano board. Connected directly to the DE0-Nano, is a 5 megapixel D5M camera module. This provides a very compact, yet flexible development platform for prototyping our design.

Configuring the camera in skipping mode, we can output a VGA resolution image (640×480) at 127 frames per second. The pixels are streamed from the camera module at 96 MHz, and the FPGA is able to maintain a throughput of 1 pixel per clock cycle at this rate. The Bayer demosaicing filter has a latency of just over 1 row (using edge directed bilinear interpolation requires a 3×3 window). Edge enhancement and noise filtering similarly use 3×3 windows, with a latency of just over 1 row for each of those operations. Colour conversion and thresholding have a latency of one clock cycle each, and run length coding provides the run of coloured pixels one clock cycle after the end of the run. Connected components detection operates on runs of coloured pixels at a time. It accumulates the area and centre of gravity of the blob as the runs

arrive. However, it must wait until the row after the end of the blob to determine that the region is completed. This adds between 1 and 2 rows latency. Distortion correction takes only a few clock cycles, as does associating the blob to a robot. Once the last coloured patch has been detected for a robot, the position and orientation can be determined, and can be output to the strategy processor. The total latency is just over 5 image rows, with the results for an object output approximately 85 μs after the last pixel for the object is sent from the camera. Note that the last object is output before the end of the frame has been reached.

5 Discussion and Conclusions

In moving the vision system from a conventional camera and video processor to an intelligent camera, we have achieved a number of benefits. Processing pixels directly at the rate provided by the sensor can maximize the resolution-frame rate product. This allows an increase in either the resolution, or frame rate, or both, over a conventional camera. Being able to directly control the sensor features also gives increased flexibility, because in a conventional camera, many of the low level features are not user accessible. Processing the data with a synchronous streamed pipeline processor also minimizes the latency. On an FPGA, each stage of the processing pipeline is built with separate hardware, so all can operate in parallel. This allows the clock rate of the system to be reduced to the native rate of the pixels being streamed from the camera. Careful transformation of the algorithm allows the objects to be identified and tracked, even before the frame has completed loading into the FPGA. On a conventional system, the video processor will not have started processing the frame yet. The higher frame rate, combined with reduced latency, can significantly improve the system controllability.

The disadvantage of an intelligent camera is the difficulty in transforming the serial, memory based, software algorithm into one suitable for synchronous streamed pipeline processing. Simply porting the algorithm generally gives disappointing results, because it is usually necessary to modify the algorithm to fully exploit the parallelism available on an FPGA.

Further enhancements which we plan to introduce within the camera are:

- Use wireless communication to transmit object data to the strategy processor. This will simplify setup and reduce the need for wires.
- Integrate calibration within the camera as outlined in (Bailey and Sen Gupta 2010). The full sensor resolution could be used to give greater accuracy for the calibration. This would significantly reduce setup time.
- Modify the colour thresholding to use adaptive thresholding rather than fixed thresholds. This would overcome the perennial problem of sensitivity to light and light distribution. Again it would reduce setup time by avoiding the need to determine suitable thresholds each time the system is set up.

Overall, we have demonstrated the significant benefits that can be obtained through using FPGAs to create an intelligent camera for object identification and tracking within the robot soccer environment. We have moved the processing burden from

computer for performing the strategy and control processing. This has enabled a significant decrease in latency (to approximately 85 μs) and increase in frame rate (to 127 fps) for VGA resolution images. We anticipate that this would lead to better control.

Acknowledgements

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

References

Appiah K, Hunter A, Dickenson P, and Owens J (2008) A run-length based connected component algorithm for FPGA implementation. In: International Conference on Field Programmable Technology; Taipei, Taiwan, pp 177-184.

Bailey D and Sen Gupta G (2004) Error assessment of robot soccer imaging system. In: Image and Vision Computing New Zealand (IVCNZ'04); Akaroa, New Zealand, pp 119-124.

Bailey DG and Johnston CT (2007) Single pass connected components analysis. In: Image and Vision Computing New Zealand (IVCNZ); Hamilton, New Zealand, pp 282-287.

Bailey DG and Sen Gupta G (2010) Automated camera calibration for robot soccer. In Robot Soccer, V. Papic, Editor. In-Tech: Vukovar, Croatia, pp 311-336.

Bailey DG (2011a) Design for embedded image processing on FPGAs. John Wiley & Sons (Asia) Pte. Ltd.: Singapore.

Bailey DG (2011b) Invited paper: Adapting algorithms for hardware implementation. In: 7th IEEE Workshop on Embedded Computer Vision; Colorado Springs, Colorado, USA, pp 177-184.

Ball DM, Wyeth GF, and Nuske S (2004) A global vision system for a robot soccer team. In: 2004 Australasian Conference on Robotics and Automation; Canberra, pp 1-7.

Bramberger M, Brunner J, Rinner B, and Schwabach H (2004) Real-time video analysis on an embedded smart camera for traffic surveillance. In: 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2004); Toronto, Canada, pp 174-181.

Bramberger M, Doblander A, Maier A, Rinner B, and Schwabach H (2006) Distributed embedded smart cameras for surveillance applications. IEEE Computer 39(2): 68-75.

Dias F, Berry F, Serot J, and Marmoiton F (2007) 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.

Downton A and Crookes D (1998) Parallel architectures for image processing. IEE Electronics & Communication Engineering Journal 10(3): 139-151.

Fatemi H (2007) Processor architecture design for smart cameras. PhD Thesis, Technische Universiteit Eindhoven

Johnston CT, Bailey DG, and Gribbon KT (2005) Optimisation of a colour segmentation and tracking algorithm for real-time FPGA implementation. In: Image and Vision Computing New Zealand (IVCNZ'05); Dunedin, New Zealand, pp 422-427.

Kleihorst RP, Abbo AA, van der Avoird A, Op de Beeck MJR, Sevat L, Wielage P, van Veen R, and van Herten H (2001) Xetal: a low-power high-performance smart camera processor. In: IEEE International Symposium on Circuits and Systems (ISCAS 2001); Sydney, Australia, vol. 5, pp 215-218.

Leeser M, Miller S, and Yu H (2004) Smart camera based on reconfigurable hardware enables diverse real-time applications. In: 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2004); Napa, California, USA, pp 147-155.

Ma N, Bailey D, and Johnston C (2008) Optimised single pass connected components analysis. In: International Conference on Field Programmable Technology; Taipei, Taiwan, pp 185-192.

Novak G and Mahlknecht S (2005) TINYPHOON: a tiny autonomous mobile robot. In: ISIE 2005; Dubrovnik, Croatia, vol. 4, pp 1533-1538.

Sen Gupta G, Bailey D, and Messom C (2004) A new colour-space for efficient and robust segmentation. In: Image and Vision Computing New Zealand (IVCNZ'04); Akaroa, New Zealand, pp 315-320.

Sen Gupta G, Messom CH, and Demidenko S (2005) Real-time identification and predictive control of fast mobile robots using global vision sensing. IEEE Transactions on Instrumentation and Measurement 54(1): 200-214.

Weiss N and Hildebrand L (2004) An exemplary robot soccer vision system. In: CLAWAR/EURON Workshop on Robots in Entertainment, Leisure and Hobby; Vienna Austria.

Weiss N and Jesse N (2004) Towards local vision in centralized robot soccer leagues: A robust and flexible vision system also allowing varying degrees of robot autonomy. In: FIRA World Congress 2004; Busan, Korea.

Wills P (1999) The hardware design of a smart camera for the robot soccer environment. BE (Hons) Thesis, Department of Computer Science and Electrical Engineering, University of Queensland.