FPGA Implementation of Hot Spot Detection in Infrared Video

Robert Walczyk[†], Alistair Armitage[*], T. David Binnie[**]

School of Engineering and the Built Environment
Edinburgh Napier University
[†]r.walczyk@napier.ac.uk
[**]td.binnie@napier.ac.uk

School of Computing
Edinburgh Napier University
[*]a.armitage@napier.ac.uk

Abstract — **This paper describes a Hardware Description Language (HDL) based fully customizable module for real-time Infrared (IR) hot spot detection and feature extraction from a video stream. The aim of the research was to investigate and evaluate possible solutions for object detection using connected component labelling that could be implemented within a streaming video embedded processing platform as a hardware accelerator. The proposed algorithm is based on a single-pass approach; this guarantees real-time processing together with very low resource utilisation. The hardware implementation was verified on a Xilinx XUP V2P (XC2VP30 FPGA) development board with an IR camera module interfaced as a real-time video source. The system was tested with an image resolution of 640×480 processing input data at a speed of 30fps which was limited by the bandwidth of the camera.**

Keywords — **FPGA, Infrared, Object Detection, Labelling.**

I Introduction

The availability of low cost thermal infrared cameras increases the viability of their inclusion in automated pedestrian detection and tracking systems. Conventional vision processing systems require fast processing and memory access as they struggle to cope with variation in lighting, shadows, reflections and other ambient image conditions. In contrast infrared images, although lower resolution, largely contain emitted radiation from hot bodies which is more tolerant to changes in ambient conditions. Infrared image processing does however have to deal with other issues such as slow variations in background temperature, clothing, and lower signal to noise. In this paper we examine some of the unique aspects of processing infrared images as we move toward the hardware implementation of a real-time pedestrian detection and tracking system.

Specifically, this paper reports the investigation into and evaluation of a hardware implementation of a Connected Component Labelling (CCL) algorithm that can be used for hot spot (object) detection and feature extraction within an infrared image processing system. CCL algorithms operate on binary images, thus the raw (grey scale) image is reduced to a binary image after noise reduction and thresholding operations prior to CCL. The traditional Connected Component Labelling approach is to first scan the binary image frame and assign each pixel with a preliminary label. A second image scan is then used to ensure all the connected pixels within component are assigned with the same label. Once the components are labeled, feature extraction can be applied once all the objects are assigned. This paper describes a technique where all the objects are detected and their features are extracted in a single image scan.

The customisable hardware available using Field Programmable Gate Array (FPGA) technology allows for concurrent processing on a parallel architecture. This means that FPGAs are an excellent means for the implementation of image processing algorithms. FPGAs can be disadvantaged by limited on-board memory leading to longer processing times if external memory has to be accessed.

In this paper a description of the very fast and

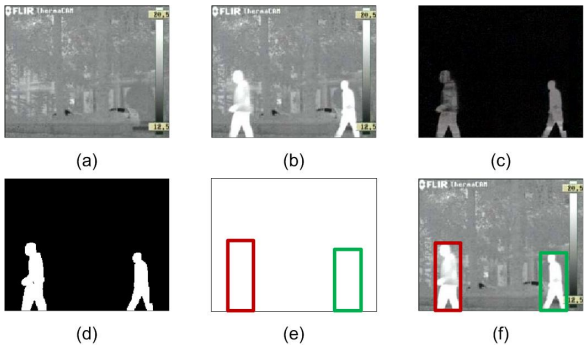

Fig. 1: Surveillance system data flow. (a) background model. (b) current frame. (c) background removal. (d) intensity thresholding. (e) object detection. (f) output image frame.

memory efficient technique for object detection and feature extraction from infrared video streams is given. This particular implementation is intended for FPGA based processing platforms and was designed as a self contained processing unit that could be applied to video processing systems as a hardware accelerator for object detection.

The outline of this paper is as follows: Section II introduces the reader with common algorithms for the CCL and feature extraction. Memory requirements are provided and a comparison of the algorithms is made. Section III gives a detailed description of the single pass algorithm developed for hardware implementation as standalone fully customizable module. The following Section IV gives implementation details with synthesis results and experimental results. Section V formulates conclusions and plans for future work.

II Detection Algorithms

An automated surveillance system for pedestrian detection with an infrared camera as a video source was described in [1]. The system digitizes video stream provided by an IR camera module and processes input data as follows: update the adaptive background model, subtract input image in order to extract regions of interest (ROIs), perform intensity thresholding and finally detect infrared hot spots (pedestrians). A block diagram of the data flow can be seen in Figure 1.

a) Labelling Algorithms

Object detection algorithms, often referred to as Connected Component Labelling (CCL) algorithms, can be categorised according to whether they scan the entire image to locate components, or trace the components contour or process a number of pixels at a time in parallel. The scan algorithms can be further classified by the number of scans that is needed for each image. They differ in execution time as well as in memory requirements. The following subsections briefly describe the main algorithms, identifying the one most suitable for hardware implementation.

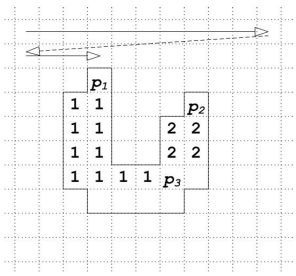

Fig. 2: A typical label collision

Most CCL algorithms classify groups of connected pixels (connected component) as disjoint objects with unique identifiers (labels). This operation can be described as assigning a unique label l taken from a set of integral values $L \subset \mathbb{N}$, to each connected component. Thereby an input binary image frame $B \in \mathbb{Z}^2$, where all the pixels $p \in B$ correspond to the background or to the foreground objects ($F_b = 0$ or $F_f = 1$ respectively), is transformed into a frame where each pixel is represented by a decimal value (label) which is the identifier of the connected component CC_k it belongs to. Here $1 \leq k \leq K$ and K defines the total number of connected components within the frame. Labelling of B can be written as $g : B \mapsto \mathbb{N}$, where $g(x, y)$ is described as:

$$g(x,y) = \begin{cases} F_b & \text{if } B(x,y) = F_b, \\ l_k & \text{if } B(x,y) \in CC_k. \end{cases} \quad (1)$$

Object features such as position, size, etc. cannot be extracted until the image is labeled completely. Also, the common problem for most of the labelling algorithms are label collisions caused by the "u" shaped objects, depicted in Figure 2. Due to the raster scanning nature of the image in digital systems, pixels are labeled on the basis of local information (adjacent pixels). Both pixels p_1 and p_2 have no direct neighbours. On the first scan, there is no information that pixels p_1 and p_2 belong to one object: they will be assigned new labels. Once p_3 is encountered, labels 2 and 1 need to be merged and previously labeled pixels need to be reassigned with a unique label for the whole object. This is the reason why most labelling algorithms need to scan the image frame more than once. Further subsections describe also how these algorithms deal with the label collision problem.

Two Pass (Classical) Algorithm

This algorithm, introduced by Rosenfeld and Pflaz [2], uses two scans through the image frame in order to label all the objects. The main disadvantage is the memory consumption (which grows rapidly

with increasing image complexity) and the need for an auxiliary memory to store all the label equivalences. It has been successfully implemented in Handel-C [3].

Multiple Scan Algorithm
Haralick presented an algorithm [4], which does not require storage for label equivalences. Ambiguities caused by label collisions are resolved on a local neighbourhood basis during iterative forward and backward image scans. This algorithm was popular for hardware implementations because of the locality of calculations and the low memory requirements [5, 6]. However, due to the number of multiple scans varying on image complexity, it cannot be employed for real-time video processing applications.

Parallel Processing Algorithm
Algorithms from this group are designed for specialised parallel processing platforms. They often use one logical processing element per pixel. Although these algorithms can be implemented on FPGAs, they require a great amount of logic resources [7], even for low resolution images, and are not efficient for labelling streaming video signals.

Contour Tracing Algorithm
Chang and Chen [8] developed a new algorithm based on contour tracing that uses a single pass through the image frame to label all the objects. It was proved that this gives better performance and uses less memory. Since objects are labeled while their contour is traced, there are no label collisions and memory is not needed for label equivalences. However, because features can still be extracted, this algorithm works well for applications which do not need uniquely labeled object masks. FPGA implementation requires two scans through the buffered image frame, and there is a slight increase in hardware complexity compared with the classical algorithm, but it is an interesting alternative. A successful implementation can be found in [9].

Single Pass Algorithm
Single pass labelling is a new approach which does not label images in the traditional manner where an output frame is created [10]. Instead, objects are detected and their features are extracted during the image scan. The main advantage of this algorithm is it does not require an input image to be buffered, and it does not use any auxiliary memory to store labeled objects. In order to perform object detection, a single row buffer has to be employed together with a line buffer and data table where object features are stored. The single scan ensures real-time processing together with very low memory requirements. was described in [11].

Tab. 1: Memory requirements for labelling algorithms with up to 255 objects per image

Resolution [pixels]	Classical [K bits]	Contour [K bits]	Single Pass [K bits]
320×240	1 280	158	12
640×480	3 355	619	18
1024×768	7 668	1 578	24

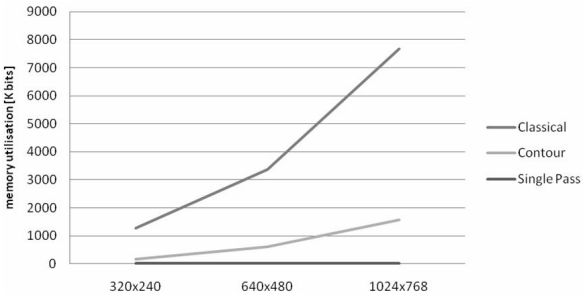

Fig. 3: Memory requirements for labelling algorithms with up to 255 objects per image

b) Summary

There is a wide range of CCL algorithms suitable for hardware implementation. Only the classical, contour tracing and single-pass algorithms can ensure real-time video processing due to constant processing time per frame. The first two algorithms require an input image frame to be buffered, and an auxiliary memory for storing labelling results in order to extract features. Assuming that every object encounters one label collision, the total amount of required memory for three different image sizes with up to 255 objects per image was calculated and can be found in Table 1; Figure 3 gives a graphical representation.

As can be seen, the single-pass algorithm outperforms other algorithms with the lowest memory requirements, it also processes an image frame at least twice faster that other algorithms.

III Infrared Spot Detection Algorithm

The IR spot detection algorithm proposed in this paper is based on a single-pass CCL. During the initial segmentation, ROIs (here hot spots) are extracted from the input image. This data is provided to the detection unit accordingly to raster scan on the pixel basis, where values 0 and 1 refer to the background and foreground objects respectively. The aim of the detection module is to distinguish disjoint groups of connected pixels as separate objects and calculate their features of interests such as position, size, bounding box and centre of gravity simultaneously while input data is provided to the system. This approach avoids the need for buffering the input image and producing an output frame with labeled objects,

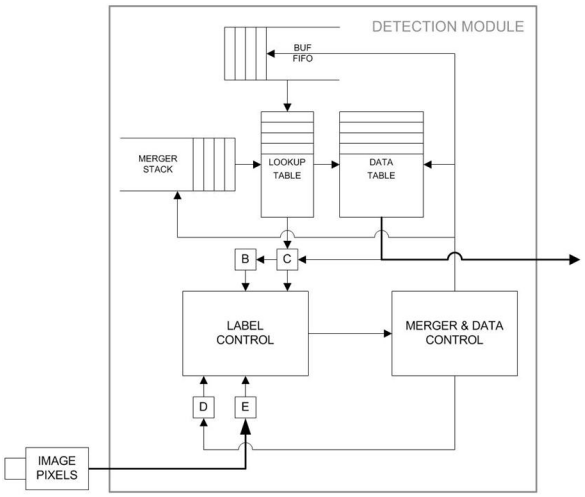

Fig. 4: Architecture of the detection module

hence the memory requirements are significantly reduced. Since object features are extracted during the single scan, this approach provides results at least twice faster than other algorithms.

a) Architecture

In order to separate various objects in a single pass, a multiple memory unit and fully pipelined data flow architecture is required. The block diagram of the detection module is depicted in Figure 4.

The detection module employs two control units: `LABEL CONTROL` and `MERGER & DATA CONTROL`, also four separate memory arrays: `BUF FIFO`, `LOOKUP TABLE`, `DATA TABLE` and `MERGER STACK`.

- `LABEL CONTROL` keeps information about two adjacent pixels to the input pixel denoted by `E`. Both pixels at `B` and `D` are analysed together with `E` in order to calculate preliminary label for the current pixel. The identifier is assigned according to the classical CCL algorithm:
  (a) do nothing when all the adjacent pixels are background pixels;
  (b) if only input pixel was found as foreground element `E`=1, assign a new label;
  (c) if only one from the neighbouring pixels was already labeled and `E`=1, assign its identifier to the current pixel;
  (d) if both adjacent pixels were already labeled with different identifiers and `E`=1, assign `E` with lower identifier and forward both labels to the subsequent control unit for further analysis;
- `MERGER & DATA CONTROL` manages label merging and feature extraction; this unit controls all the memory modules within the detection unit;
- `BUF FIFO` is a fifo buffer that stores labels for each pixel in the previous row; already read labels are 'overwritten' with the data from current row;
- `LOOKUP TABLE` is a translation table that gives a pointer to the particular label (label equivalence) read from the `BUF FIFO`; all the newly created labels are pointing to themselves, this particular memory module makes sure that previously merged labels use current identifiers;
- `DATA TABLE` stores extracted features for all the detected objects;
- `MERGER STACK` this particular module keeps pairs of labels that need to be merged;

b) Object Detection

Let us consider the binary image depicted in Figure 5. During the first (`I`) line scan, the `BUF` row buffer is filled with 0s. Once the pixel p_1 is encountered, `BUF` is updated, a new entry in the `LOOKUP` is created for label 1 pointing to itself, also the `DATA` is updated with newly calculated features for the object 1. This procedure is repeated for pixels p_2 and p_3 with labels 2 and 3 respectively. Once the pixel p_4 is encountered, both labels 3 and 2 need to be merged. A lower index (2) is chosen, all the features for labels 2 and 3 stored in `DATA` are combined and the `DATA` is updated at 2, position 3 is cleared. The `LOOKUP` is also immediately updated with label 2 at location 3. However, there is a risk that a series of mergers will occur, which may lead to the situation where cells within the `LOOKUP` will be pointing to already dead labels. To ensure the `LOOKUP` is updated with recent labels before the next line scan, both labels 3 and 2 are pushed onto the `MERGER STACK`. A similar situation occurs at pixel p_5. `LOOKUP` is immediately updated at index 2 with label 1, the `DATA` at position 1 is overwritten with newly calculated features, labels 2 and 1 are pushed onto the stack. Since `MERGER STACK` is not empty at the end of the `IV` line scan, `LOOKUP` is revised with label pairs popped from the stack in the reverse order than they were pushed onto the stack: 2 and 1, 3 and 2 respectively. As the `LOOKUP` was updated with recent labels before the `V` line scan (values at the left of /), pixel p_6 is assigned with label 1 and features for this label are updated. Otherwise (values at the right of /), label 2 would be assigned with features calculated for this pixel only.

c) Feature Extraction

Within the `MERGER & DATA CONTROL` unit the detection module extracts features such as location, size, coordinates of the bounding box and its centre of gravity. The area is calculated as a sum of foreground pixels $I(x, y) = 1$ for each object. The bounding box gives coordinates of the smallest rectangle containing the detected object. The lo-

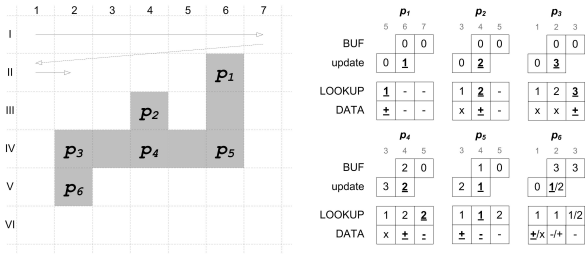

Fig. 5: Object detection algorithm; a). binary input image; b). memory registers, where values recently changed are underlined, "-" indicates empty cell, "x" previously assigned value, "+" stands for an update.

cation is defined as the intersection of diagonals of the bounding box. The Centre of Gravity (CoG), also referred to as centroid gives, coordinates of the centre of mass and can be calculated according to:

$$\overline{x} = \frac{M_{10}}{M_{00}}, \quad \overline{y} = \frac{M_{01}}{M_{00}}, \tag{2}$$

where image moments M_{10}, M_{01} and M_{00} can be obtained using:

$$M_{ij} = \sum_{x} \sum_{y} x^i y^i I(x, y). \tag{3}$$

IV Hardware Implementation

During the initial development stage, the code was written and tested using Matlab environment. A fully featured simulation model was created where the algorithm was verified. An additional script was created to generate a VHDL ROM file with a test image pattern for visual verification of the hardware implementation. Results of the processing were verified with red bounding boxes overlapping detected objects. Extracted features were compared with simulation results using internal and external logic analysers, Chip Scope Pro (CSP) and Ant16 respectively.

The main advantage of the presented detection module is the fact it was fully designed using VHDL and all the memory blocks used within the design were instantiated using behavioural HDL inference template. This method is semi-device independent and allows the designer to easily change memory size with the use of generic parameters.

a) Module Instantiation

The detection module can be customized using generic parameters that specify maximum number of objects per frame, define width of the label counter, also size of the input image. These generics will be used by synthesis tool to adjust sizes for memory units and registers to particular video signal.

Together with pixel value register `E`, there are also pixel clock and reset signal included within the entity, also horizontal and vertical counter that give current location of the pixel.

There are two output signals supported by detection module: data vector which provides all the calculated features and boolean signal specifying whether the current pixel should be displayed as bounding box or not.

b) Object Detection Mapping

Input data pixel is stored in the register `E`. After a clock cycle this value is shifted in a window filter manner to the register `D` being direct left neighbour. The top adjacent pixel stored in the register `B` is decoded from the `LOOKUP` which is accessed at the address that is provided by the output register from `BUF`. Since majority of the memory modules within the design are synthesized as BlockRAM, this requires that `BUF` to be accessed for two clock cycles in advance. Object detection and feature extraction are performed on the basis of the local neighbourhood of the pixel `E` and data stored in the registers regarding these pixels. To reduce the number of memory accesses, temporary values are stored in the registers. The `DATA` is updated once the end of the object was detected (`E`=0, `D`=1). The `BUF` keeps recording labels from the current location, whereas `LOOKUP` is updated only when a new object is encountered or two labels are merged. The `LOOKUP` write enable signal is also multiplexed with control signal from the stack-based merging process. This process runs only when the `MERGER STACK` is not empty and is completed during the horizontal blanking period. At the end of the image scan after the calculated data was passed for further processing, all the memory modules are cleared to ensure no error will occur in the next image scan.

c) Feature Extraction Mapping

The main assumption for described detection module was it will be capable of extracting the following features: position, size, bounding box, CoG. In order to reduce storage memory, position of the detected object is not stored in the memory as is can be calculated online based on the coordinates defining bounding box (as average for both minimum and maximum x and y coordinates). Bounding box is stored in the memory as coordinates of the top-left and bottom-right corner which is updated while the object grows. The size of the object was calculated as a sum of pixels `E`=1. Once two labels were merged, the size was calculated as a total sum of both sums. The most difficult feature to extract is CoG. It is calculated as two image moment M_{10} and M_{01} divided by the M_{00}, where M_{00} is defined by the area (size) of the object accord-

Fig. 6: Detection results

ing to equation (2). The M_{10} was implemented as a sum of column numbers and the M_{01} as a sum of row numbers for pixels `E`=1 within particular object. The division was implemented using the Pipelined Divider 3.0 from Xilinx Core Generator.

d) Experimental Results

The implementation of the detection module gave excellent performance; results can be seen in Figure 6. The processing was carried out in real-time at the frame rate of the video source with a resolution of 640 × 480 pixels. The report from the synthesis tool for basic implementation (bounding boxes only with up to 255 objects per frame) using XC2VP30 FPGA stated less than 2% resource utilisation with only 4 BlockRAMs used. Implementation with CoG extraction causes little increase in hardware complexity and the number of BlockRAMs used highly depends on the precision of the division (fractional remainder width up to 32 bits).

Since the divider module was generated using Core Generator, for optimal resource utilisation it has to be re-generated when the generic parameters are changed. The implementation of a custom divider module is currently being investigated.

In order to further reduce memory requirements, a stack-based label counter was implemented. Once two labels are merged, the `LOOKUP` is updated as described previously, the `DATA` is also cleared for the higher label. The proposed feature ensures that no memory is wasted by these 'dead labels'. Once a new object is encountered, it is assigned with a label from the label-stack if not empty, otherwise the one from the label counter is used.

V Conclusions

In this paper we have described the design of a fully customizable generic HDL module for hot spot detection and feature extraction from an infrared video stream. This work has shown that it is possible to detect hot spots (people) in IR video sequences at the full frame rate. The key to this is the use of a single-pass algorithm for the component labelling stage that follows the relatively straightforward separation of the image into foreground and background based on temperature.

The FPGA implementation of the algorithm works well, and is able to label hot spots, as well as locate them, produce a bounding box around them, and their centre of gravity, all in a single image scan. People tend to present a non-uniform shape, and in locating such shapes, the CoG is a useful measure for tracking algorithms. The system currently produces this value, but the implementation is sub-optimal, and we are currently working on improving this as we move toward the hardware implementation of a real-time pedestrian detection and tracking system.

References

[1] R. Walczyk, A. Armitage, and T.D. Binnie. An Embedded Real-Time Pedestrian Detection System Using an Infrared Camera. In *IET Irish Signals and Systems Conference, 2009. IET ISSC 2009.*

[2] A. Rosenfeld and J.L. Pfaltz. Sequential operations in digital picture processing. *J. ACM*, 13(4):471–494, 1966.

[3] M. Jablonski and M. Gorgon. Handel-C implementation of classical component labelling algorithm. In *Digital System Design, 2004. DSD 2004. Euromicro Symposium on*, pages 387–393.

[4] R.M. Haralick. Some neighborhood operations. *Real Time/Parallel Computing Image Analysis*, pages 11–35, 1981.

[5] D. Crookes and K. Benkrid. FPGA implementation of image component labelling. *Reconfigurable Technology: FPGAs for Computing and Applications*, 1999.

[6] K. Appiah and A. Hunter. A single-chip FPGA implementation of real-time adaptive background model. In *2005 IEEE International Conference on Field-Programmable Technology, 2005. Proceedings*, pages 95–102, 2005.

[7] E. Mozef, S. Weber, J. Jaber, and G. Prieur. Parallel architecture dedicated to image component labeling in O (n Log n): FPGA implementation. In *Proceedings of SPIE*, volume 2784, page 120, 1996.

[8] F. Chang, C.J. Chen, and C.J. Lu. A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, 93(2):206–220, 2004.

[9] H. Hedberg, F. Kristensen, and V. Owall. Implementation of a labeling algorithm based on contour tracing with feature extraction. In *IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007*, pages 1101–1104, 2007.

[10] D.G. Bailey and C.T. Johnston. Single pass connected components analysis. In *Image and Vision Computing New Zealand*, pages 282–287, 2008.

[11] C.T. Johnston and D.G. Bailey. FPGA implementation of a single pass connected components algorithm. *Electronic Design, Test and Applications*, pages 228–231, 2008.