FPGA IMPLEMENTATION FOR FAST INFRARED SPOTS DETECTION

Maoxiang Huang, Chenhao Wang and Yuncai Liu

Institute of Image Processing and Pattern Recognition
Shanghai Jiao Tong University, China
Email: maoxianghuang@sjtu.edu.cn

ABSTRACT

This paper introduces a fast infrared spots detection algorithm designed for field-programmable gate array (FPGA) implementation. The proposed algorithm processes four pixels per clock cycle and detects infrared spots in a single pass over a frame. The implementation of the algorithm is only composed of combinatorial logic and registers. Furthermore, the execution time of the algorithm is independent of image content. For prototyping and evaluation purposes, the algorithm is implemented in an FPGA device. Demonstrated its superiority over the existing multi-pass algorithms and some other one-pass algorithms, it processes 1024×768 images smoothly at 60 fps and detects infrared spots in a 1024×768 image within 1.966ms.

Index Terms— Infrared spots detection, Real-time image processing, Parallel architecture, Centroid calculation

1. INTRODUCTION

Infrared spots detection, the task of finding and locating infrared spots in images, is the key technique used in optical measure instruments, especially in infrared surgical navigation system which is very strict about real-time and accurate performance. It is well know that FPGA has significant advantages in real-time parallel image processing, especially in low levels vision processing for huge amounts of data, such as binarization and connected component detection. In recent years, much research has been carried out on real-time image processing by means of parallel architecture (e.g. [1,2,3,4]).

In the field of pattern recognition, the detection of connected components is an essential task that finds the features of objects (e.g. [1]). For real-time image processing, many connected component labeling algorithms and corresponding parallel hardware architectures have been investigated. The classical two-pass algorithm proposed by Rosenfeld and Pfalz [5], performing in a raster scan order, is suitable for hardware implementation. However sequential scanning of an image requires excessive time consumption, and a large quantity of equivalence label pairs occupies high dynamic storage. To reduce the massive storage requirement, Haralick [6] introduced an iteration scheme by scanning an image repeatedly. Yet, the algorithm consumes more time for labeling computation. Subsequently contour tracing algorithm [e.g. 4] was proposed to ameliorate the storage space and computation time. This algorithm completes labeling in a single scan, but it calls for accessing the image memory in irregular mode which can’t adapt to hardware implementation. In addition, parallel hardware architecture was presented by Wang et al. [2] at the cost of high hardware logic resources. In this algorithm, even though two parallel hardware processing units are employed, the execution time cannot be reduced remarkably since the pixel is not transferred in parallel. Amir et al. [1] successfully implemented an algorithm for eye detection based on FPGA, including computation of regions, shape moments and simple shape classification, all of which are computed within a single pass over the frame. The algorithm also doesn’t execute at a high enough speed, because it processes one pixel per clock. When several connected components need merging, it is required to find the lowest component identification number (ID), so that unnecessary latency and hardware logic are increased. Additionally, there is a possibility that the line component which contains the last pixel of the line can’t be merged with components of previous lines to form connected regions, as the connect components module proposed in [1] is reset every line clock.

In this paper, we propose a fast infrared spots detection algorithm for hardware implementation on FPGA. This algorithm uses 8-adjacency to detect the all neighbors’ components and computes the infrared spots in a single pass over the frame. If the pixel depth is 8 bits, it processes four pixels per clock with regular memory access. The processing time is much less than the multi-pass algorithms (e.g. [2,5,6]), and at least three times less than some other one-pass algorithms (e.g. [1,3]). The implementation has low memory requirements and can detect infrared spots anywhere in an image. It surmounts most of the storage bottlenecks presented in the previous applications, and also overcomes the disadvantages of [1]. Based on system on programmable chip (SOPC) approach, the infrared spots detection unit is seamlessly integrated into the module proposed. The following sections discuss the design details.

This work is supported by China National 973 Program (2006CB303103), China NSFC Key Program (60833009), National 863 Program (2009AA01Z330) and Shanghai Municipal Natural Science Foundation (08ZR1410700).
2. INFRARED SPOTS DETECTION ALGORITHM

The algorithm for infrared spots detection includes binarizing image to obtain regions image, finding regions (connected components) of infrared spots and computing each region’s shape properties and centroid. For the sake of fully exploiting the concurrency of hardware implementations, the algorithm is performed in parallel using pipelined processing blocks. Fig. 1 shows a block diagram of the parallel algorithm.

2.1. FIFO buffer and binarization

FIFO buffer receives the images from memory buffer with regular access pattern. In current implementation, 32-bit FIFO buffer is selected, and therefore one line of a 1024 × 768 image frame (8 bits per pixel) is transferred within 256 clock cycles. Additionally, in order to acquire the row number and column number of the inputted pixel, the FIFO buffer produces a valid data signal to activate two counters that are responsible for counting the inputted pixels.

Since the infrared spots are bright enough, it is easy to choose a binarization threshold. After binarization, the ‘1’ pixel is the object (connected component) pixel and the ‘0’ pixel is the background pixel.

2.2. Detecting line components and their properties

Once the row counter increases by 1, this module is triggered. 4-bit data resulting from binarization unit are inputted for detecting line components every one clock; accordingly, there will be 16 kinds of possibilities per clock for line components computation. The starting pixel \( (c_{\text{start}}) \), the ending pixel \( (c_{\text{end}}) \), and first-order moments \( (M_{00}, M_{10}, M_{01}) \) are calculated for each line component. Where \( M_{00} \) is the number of pixels in line component, \( M_{01} \) is the sum of column numbers in line component, \( M_{10} \) is equal to the product of \( M_{00} \) and row number of the line component. All of these properties are stored in the embedded RAM of FPGA. For an image size of 1024 × 768, the memory arrays that store the properties with the corresponding bits-width are: \( c_{\text{start}} \) (10 bits), \( c_{\text{end}} \) (10 bits), \( M_{00} \) (8 bits), \( M_{01} \) (24 bits), and \( M_{10} \) (24 bits).

2.3. Merging line components and maintaining regions list

The connecting components module implements on two consecutive lines of an image frame. It fetches \( c_{\text{start}} \) and \( c_{\text{end}} \) arrays of current line from the previous module and accesses \( c_{\text{start}} \) and \( c_{\text{end}} \) arrays of previous line from RAM. Besides, it labels every component with a component ID and maintains regions list. Additionally, four properties, that is, \( C_{L}^\text{min} \), \( C_{L}^\text{max} \), \( R_{L}^{\text{min}} \), and \( R_{L}^{\text{max}} \) are calculated in this module, e.g.
\[
C_{L}^\text{min} = \begin{cases} 
C_{L}^\text{min} & \text{if } C_{L}^\text{min} < C_{L}^\text{max} \\
C_{L}^{\text{min} \cdot L} & \text{otherwise}
\end{cases}
\]

Where the subscript \( L \) refers to a region, and the subscripts \( L_1 \) and \( L_2 \) represent line components that belong to \( L \). The region \( L \) is discovered in line-by-line scan order and consists of multiple line components. After one-pass scanning for a frame, \( C_{L}^\text{min} \) is the minimum column number of pixel in the region \( L \).

Other detailed implementation of this module is reported in [1]. Being different from [1], this algorithm uses 8-adjacency to find the regions. The following guideline is used to detect all (eight-) neighbors’ line components:
\[
(C_{\text{start}} \leq c_{\text{end}}) \text{ and } (c_{\text{start}} \leq c_{\text{end}}^p)
\]

Where,
\[
c_{\text{end}}^p = \begin{cases} 
C_{\text{end}} + 1, & \text{if } 1 \leq C_{\text{end}} < C_{\text{num}} \\
C_{\text{num}}, & \text{otherwise}
\end{cases}
\]
\[
c_{\text{start}}^p = \begin{cases} 
C_{\text{start}} - 1, & \text{if } 1 < C_{\text{start}} \leq C_{\text{num}} \\
1, & \text{otherwise}
\end{cases}
\]
\( C_{\text{start}} \) is the column number of the starting pixel in current line component. \( C_{\text{end}} \) is the column number of the ending pixel in current line component. \( C_{\text{start}}^{P} \) is the column number of the starting pixel in previous line component. \( C_{\text{end}}^{P} \) is the column number of the ending pixel in previous line component. \( C_{\text{num}} \) is the number of columns in image frame.

Four cases of line components merging are shown in Fig. 2. Applying this guideline, the algorithm compares every component in the current line with all the components in the previous line. Another difference is this algorithm does not need to find the lowest ID when multiple components are being merged. It will also maintain the minimum memory size for component properties storing. At the same time, it saves the hardware logic resources and reduces the potential latency. In this algorithm, when several components are merged, their moments are added together respectively. If all of these components to be merged have not been merged before, the result is stored in the region with the component ID of the last to be merged; and if at least one of the components has been merged before, the result is stored in the region with the component ID of the last having been merged.

This algorithm reads the input image frame once with regular data access. According to [1], we can know that the number of components per line is limited to the square root of the number of pixels in a line, if one pixel is computed every clock cycle. For a 1024×768 frame size with 8-bit pixel depth, as the unit processes four pixels per clock cycle, it can support up to 16 objects per line, and only need \((256 \times 768 + 16)\) clocks for computing connected components.

Occasionally, line components are in the rightmost side of an image. They can’t be merged with components of previous lines to form connected regions, since the connect components module proposed in [1] is reset every line clock. To overcome this disadvantage, once the row counter increases by 1, this unit isn’t reset until inserting 16-clock delay.

2.4. Computing the detected regions

As soon as the regions detection of the entire frame is finished, this unit checks for the shape properties of each region and decides whether the region is a valid infrared spot. The centroid coordinates of an infrared spot are then calculated using:

\[
X = \frac{\sum M_{10}}{\sum M_{00}} \quad Y = \frac{\sum M_{01}}{\sum M_{00}}
\]

Where \( X, Y \) are the centroid coordinates of the infrared spot. The results are stored in the embedded RAM of FPGA. When the row counter’s output value is equal to the number of rows in image frame, this unit is triggered after inserting 16-clock delay. In current implementation, it copies the list of regions’ moments calculated in the previous unit, to enable processing of the next frame concurrently, and generates an interrupt to inform Nios II processor that it needs data output after the calculation of all regions’ centroids.

3. HARDWARE IMPLEMENTATION

An FPGA module was designed for hardware implementation. It uses two high-resolution IEEE 1394b cameras (Point Grey FL2-08S2M), each of which has the capability of supporting monochrome video at 1024×768 resolution up to 30 frames per second. The system architecture of the module is shown in Fig. 3. IEEE 1394b link-layer controller (TSB82A2A2) and physical layer controller (TSB81B3A3) provide the communication ports for IEEE 1394b cameras. SDRAM provides a buffer for image data. Pulse width modulation (PWM) module adjusts the brightness of LEDs for camera. And external memory interface (EMIF) and host port interface (HPI) are configured for frames and centroid coordinates output respectively. UART is an optional port which can be used for camera control. The only processing unit used for this module is an Altera Cyclone II family FPGA EP2C70.

The internal logic units of FPGA include PCI host bridge interface, SDRAM controller, PWM controller, UART controller, EMIF interface, HPI interface, infrared spots detection unit, Avalon-MM interface, on-chip RAM and Nios II processor unit all of which are seamlessly integrated into the overall system by Altera SOPC Builder tool. There is a significant difference compared to the approach of software-based implementation on PC. In this system, the Nios II processor is not the core unit of the system. The processor only carried out a few functions. It is mainly responsible for coordinating the work of other units.

4. EXPERIMENTAL RESULTS

Given an \( N \times N \) binary image, our design takes \((N^2/4 + \sqrt{N}/4)\) cycles for computing the connected components’ centroids. Table 1 lists the different computing time for centroids computation between [1] and our method. Table 2 lists the

![Fig. 3. Architecture of the FPGA module.](image-url)
differences in execution time for connected component labeling algorithms [5,6,2,3]. Obviously, those approaches [1,5,6,2,3] require more processing cycles than ours.

The dedicated printed circuit board (PCB) was designed for the prototype implementation. Using the two IEEE 1394b cameras (Point Grey FL2-08S2M), the prototype system runs at an impressive speed of 100MHz, simultaneously, transfers and processes 1024 × 768 images smoothly at 60 fps. In terms of the maximum speed as reported by the Altera TimeQuest Timing Analyzer tool, the system can run at a speed of 125.04MHz.

Compared with software-based implementation for computation of the infrared spots’ centroids, the system outputs the centroids’ coordinates with the same sub-pixel accuracy. In Fig. 4, there are 9 infrared spots, and the total computing time for this frame is (256 × 768 + 16) cycles, i.e. 1.966ms. Consequently, the system is capable of processing 1024×768 progressive scan frames at a 500 fps rate. At the same time, the result confirms that this hardware implementation of the proposed algorithm is fast enough to support the time-critical machine vision system.

Some details about the FPGA device utilization are reported in table 3. In fact, the algorithm implementation consists of combinatorial logic and registers only. It can be seen that the highest usage percentage is on the external pins, whereas there are lots of internal logic resources available for further processing blocks. In our current application, the cache memory size of Nios II processor is 6 Kbytes and On-chip RAM occupies 12 Kbytes. In brief, the implementation of this hardware architecture has low hardware resource requirements.

### Table 1. Hardware performance comparison

<table>
<thead>
<tr>
<th>Author</th>
<th>Execution time (cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amir [1]</td>
<td>$N^2 + \sqrt{N}$</td>
</tr>
<tr>
<td>Ours</td>
<td>$N^2/4 + \sqrt{N}/4$</td>
</tr>
</tbody>
</table>

### Table 2. Comparison among the different connected component labeling algorithms [5,6,2,3]

<table>
<thead>
<tr>
<th>Author</th>
<th>Execution time (cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rosenfeld [5]</td>
<td>$3N^2$</td>
</tr>
<tr>
<td>Haralick [6]</td>
<td>$2mN^2$</td>
</tr>
<tr>
<td>Wang [2]</td>
<td>$N^2 + 6N - 4$</td>
</tr>
<tr>
<td>Ito [3]</td>
<td>$N^2 + L$</td>
</tr>
</tbody>
</table>

### Table 3. Device utilization

<table>
<thead>
<tr>
<th>Resource</th>
<th>Total</th>
<th>Total Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Elements</td>
<td>68,416</td>
<td>8,578(13%)</td>
</tr>
<tr>
<td>Pins</td>
<td>422</td>
<td>159(38%)</td>
</tr>
<tr>
<td>Memory bits</td>
<td>1,152,000</td>
<td>199,026(17%)</td>
</tr>
<tr>
<td>Embedded Multiplier (9 bits)</td>
<td>300</td>
<td>0</td>
</tr>
<tr>
<td>PLL</td>
<td>4</td>
<td>1(25%)</td>
</tr>
</tbody>
</table>

5. CONCLUSIONS

In this paper, we have presented the design and implementation of an FPGA module, which performs fast infrared spots detection. To execute fast infrared spots detection, a parallel algorithm that processes four pixels per clock cycle is proposed. It uses 8-adjacency to detect all (eight-) neighbors’ components and detects infrared spots in a single pass over a frame. The implementation is only composed of combinatorial logic and registers. Furthermore, the execution time of the algorithm is independent of image content. A prototype system is implemented in an FPGA device. It is capable of transferring 1024×768 images smoothly at 60 fps and detecting infrared sports in a 1024×768 image within 1.966ms, demonstrated its superiority over the existing multi-pass algorithms and some other one-pass algorithms.

6. REFERENCES