

Efficient Image Filtering and Information Reduction in Reconfigurable Logic

Jim Torresen¹, Jorgen W. Bakke¹ and Lukas Sekanina²

¹Department of Informatics, University of Oslo P.O. Box 1080 Blindern, N-0316 Oslo, Norway

²Faculty of Information Technology, Brno University of Technology Bozetechova 2, 612 66 Brno, Czech Republic

E-mail: {jimtoer,jorgenwa}@ifi.uio.no, sekanina@fit.vutbr.cz

Abstract

An automatic sign detection system could be important in enhancing traffic safety. Such a system would have to be able to provide high speed processing in its real-time environment. In this paper, we show how one of the time consuming parts of a speed limit detection algorithm can be implemented in reconfigurable logic to speed up the processing. Results indicate that the present system would be able to handle 12 images per second.

1 Introduction

Recognizing traffic signs automatically would probably be important for traffic safety in the future. It would inform drivers about signs they did not notice before passing them. Specifically speed limit sign recognition – studied in this paper, could inform drivers about the present speed limit as well as giving an alert if a car is driven faster than allowed. In the future, autonomous vehicles would probably have to be controlled by automatic road sign recognition. Detection systems would have to provide real-time processing of images collected in a fast moving vehicle. At the same time, such a system should have a reasonable *cost* to be of interest in average cars. Thus, it is not possible to apply expensive processor-based architectures. In this paper, we introduce reconfigurable logic (by Field Programmable Gate Array – FPGA) processing for the most computational demanding parts of our speed limit sign recognition system.

Recognizing road images have been studied for a long time. The first known attempt for making a real-time system was by Akatsuka and Imai [1]. Many techniques have been proposed since then.

The standard technique for detecting and recognizing road signs consists of three steps [5]. First, color segmentation or color thresholding is applied to emphasize possible signs in an image. Thus, restricting the search area in the image. Second, template matching is applied for shape detection. Third, specific signs are detected using template matching or neural networks.

This paper concerns detection of speed limit signs specifically. We have found very little work on speed limit sign classification. There exist a system based on Global Position System (GPS) and digital road maps with speed limits included [10]. However, such a sys-

tem depend on much external infrastructure. Further, problems like lack of updated speed limits on road maps question the reliability of such a system.

Regarding hardware implementation, there is little work on reconfigurable logic applied to intelligent transportation systems. Most systems are based on software only. In the case the real-time aspect has been addressed, it has been by parallel processing systems with processors or Application Specific Integrated Circuits (ASICs) [2].

There are some work conducted on reconfigurable logic applied to image processing in general. In [6], an architecture consisting of parallel processing elements is proposed. A pixel processor for object detection applications is introduced in [7]. Substantially faster (8 to 800 times faster) image processing in FPGA than on a PC is documented in [3]. Convolution operation implemented in FPGA to be applied for real-time image processing is given in [4]. It has also been proposed to evolve image filters in reconfigurable logic [8].

Earlier we have proposed algorithms for speed limited sign detection [9]. It has later been optimized for both better performance and speed [11]. In this paper we will introduce an efficient implementation in reconfigurable logic of one of the most computational demanding parts of our system. Further, we will show how this improves the speed to make real-time processing possible.

The next section introduces our earlier work on the detecting of Norwegian speed limit signs. This is followed by an outline of the proposed FPGA implementation in Section 3 and results from the implementation in Section 4, respectively. Finally, Section 5 concludes the paper.

2 Norwegian Speed Limit Signs Detection System

Speed limit signs have features making them feasible for automatic detection. First, there is a limited number of signs to distinguish. In the experiments we have been using the following speed limit signs: 30, 40, 50, 60, 70 and 80. A typical input image is shown in Fig. 1. The outer circle of a sign is in *red* color.

The detection algorithm is divided into *three* parts: 1) Image filtering to emphasize the red parts of the



Fig. 1. An example of an input image.

sign(s), 2) Template matching to locate the sign(s) in an image and 3) Sign number recognition [11]. These will be presented in the following sections below.

Image Filtering

In this part, a specialized robust color filter is applied on the image to mark the *red circle* surrounding the speed limit numbers. It consist of two main parts. The first part – called Image filtering RWB (Red White Black), extracts the three colors red, white and black. The second part - called Red Reduction, reduces the number of red pixels in the image to minimize succeeding template matching processing. The detection algorithm has to use a broad definition of red and white colors since various intensities and saturations can occur in real images (depending on weather, day/night-light, etc.). On the other hand, only red color *on* traffic signs should be detected to minimize the number of starting points for the following template matching algorithm. Thus, detection of red color *outside* a traffic sign (like on buildings, cars, etc.) should be minimized.

The RGB color model was found useful for detection. RGB is also appropriate for hardware implementation since this will be the output from the camera. Converting into e.g. the HSV model would be computational expensive. The colors are defined (by experiments) as follows in the Image filtering RWB:

- A pixel is RED if: $(R > 77) \text{ AND } (R - G > 17) \text{ AND } (R - B > 17)$
- A pixel is WHITE if: $(R > 108) \text{ AND } (G > 108) \text{ AND } (B > 108)$
- A pixel is BLACK if: $(R < 122) \text{ AND } (G < 122) \text{ AND } (B < 122)$

RED, WHITE and BLACK are symbolic names that will be used below. As one can see, the definition of the basic colors is broad to assure robustness. WHITE and BLACK are used mainly in different parts of the algorithm. Thus, there is a value overlap (108 to 122) between BLACK and WHITE. Good selectivity is achieved by combining these colors during pixel classification. The output from filtering the image in Fig. 1 is given in Fig. 2.

The brightness varies a lot between the images. To be able to find signs in either light or dark images, the pixel values have to be made darker or lighter, respectively. This is undertaken if the average pixel values in

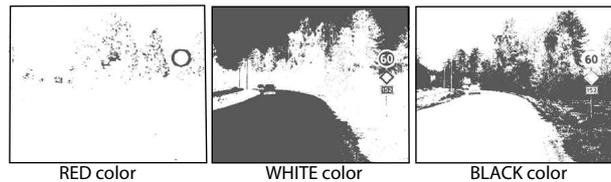


Fig. 2. The result of each of the three color filters. (Dark color mark pixels with the filtered color in each of the images.)

an image is less than 125 (too dark image) or above 150 (too light image). The pixel values are to be modified by the difference between the average pixel value and the dark or light threshold values given above. Adjusting the value of each pixels in an image adds much computation time. Thus, the *thresholds* for the colors (RED, WHITE and BLACK) are modified instead. This results in the same recognition performance but less time is needed.

The following filtering algorithm is applied to reduce the number of red pixels in Red Reduction:

1. Find all 2x2 WHITE squares in the image (since there are white pixels *inside* the red circle on the sign).
2. Find (recursively) all RED neighbors of the located WHITE squares.
3. Apply template matching (described in the next section) to the RED pixels found.

This algorithm effectively limits the number of red pixels in the image to be further processed.

Locating Sign by Template Matching

We locate signs in the image by searching for the *red circle* surrounding the numbers. The search is based on template matching. One reason for this is that template matching can be very effectively implemented in hardware. To limit the computation time, we have found that six templates are sufficient. This was based on analyzing sign sizes in images taken from a vehicle. The templates are of size 32x32, 38x38, 46x46, 52x52, 62x62 and 78x78 pixels and are used to detect the *position* of a speed limit sign. The templates are shown in Fig. 3. We need a very high reliability of the system and small templates tend to produces incorrect classification of the numbers. The template matching algorithm is described in more detail in [11]. In addition to locating the sign, the algorithm tries to reject objects that are not signs and signs that are not speed limit signs. That is, to improve recognition at the same time as reducing the computation time.

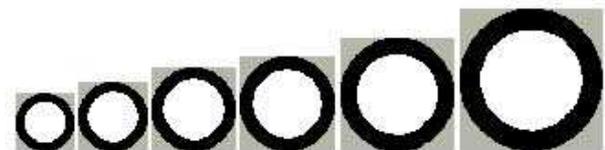


Fig. 3. Templates used for locating signs in an image.

Sign Number Recognition

The last part of the algorithm is to detect the speed limit *number* on a sign.

This is conducted as follows:

1. Clean the space defined by the best template (remove RED and surrounding colors), but keep the numbers.
2. Find boundaries of numbers (width and height).
3. Work only with the first number (the second is always zero).
4. Create a 7 (rows) x 5 (columns) bit array of the given number (down-scaling): Set each bit of the array to 1 if there is more BLACK than WHITE pixels, else set the bit to 0.
5. Classify the bit array using a classifier system.

To classify the number given by the 7 x 5 bit array, we have got a high detection rate both with a feed-forward neural network trained by the back-propagation algorithm [11] as well as classification by evolvable hardware [12]. The experiments have so far been based on single images. Thus, future work would consist of implementing a real-time prototype system. This would be required to be able to verify the system in a real environment.

Timing

The optimized algorithms performing the steps described in the previous sections require a total processing time of 130 ms per image. This corresponds to approximately 8 images per second. This is measured on a PC with a 1.4 GHz Athlon AMD processor (512Mbyte RAM). Such a fast/expensive processor could not be expected in an embedded detection system for a normal car. The processing time for each of the three steps of processing is as given in Table I.

TABLE I
THE TIME USED FOR EACH STEP OF THE SPEED LIMIT SIGN RECOGNITION.

Operation	Time (ms)
Image filtering RWB	50
Red Reducing	40
Locating sign and recognizing number	40
Total	130

The most computational demanding part is the image filtering. Filtering into RED, BLACK and WHITE colors consists of operations that can be conducted in parallel. In the next section, an implementation of these operations in reconfigurable logic will be presented. We have not yet considered implementing the Red Reduction in hardware. This will require both a complex architecture as well as being difficult to parallelize in the FPGA.

We have been careful in developing our succeeding software algorithms in such a way that they *only* depend on the RED, WHITE and BLACK *bit-level* information for each pixel. Thus, only the output of the image filtering system – consisting of three bits for each pixel, would be needed. Therefore, the original image is only needed as input to the filtering hardware and *not* by the following software system.

3 Image Filtering in Reconfigurable Logic

Having identified that image filtering is one of the computational demanding parts of the algorithm, we will in this section propose an architecture for image filtering in reconfigurable logic. A top-level view of the system is given in Fig. 4.

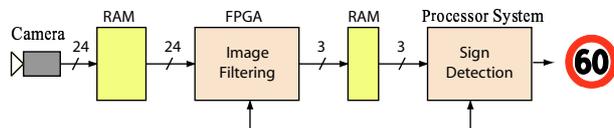


Fig. 4. A block level description of image filtering in reconfigurable logic.

The image captured by a camera is first stored in a RAM where each pixel is represented with 8 bit resolution for each of the three colors (red, green and blue). After image-filtering in the FPGA, each pixel is represented with *three* bits – one bit for each of the colors red, white and black, respectively. Thus, the data bus to the RAM used to store the FPGA output needs to be only 3 bit wide. The rest of the processing will be conducted by an embedded processor (not yet implemented). This system provides pipelined processing between the FPGA and the processor. Thus, our goal is to obtain less time being used in the FPGA than in the processor for each image.

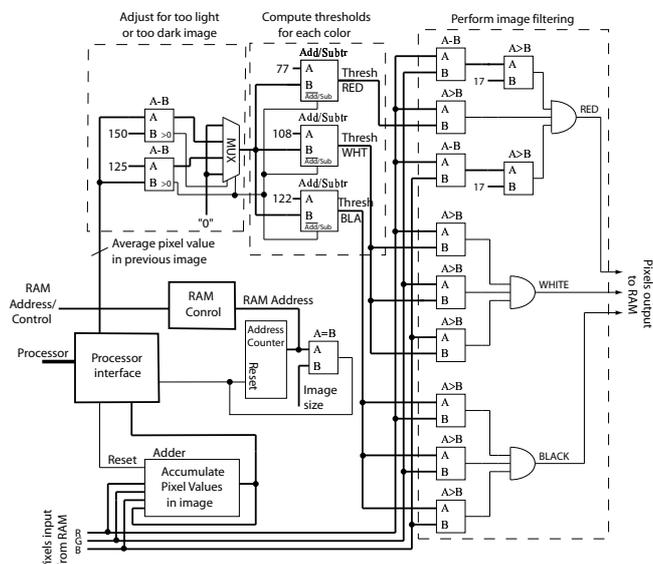


Fig. 5. The image filtering implementation in reconfigurable logic.

The details of the implementation are included in Fig. 5. Image filtering for each pixel is undertaken in a fully parallel architecture. As described in Section 2, the *thresholds* for each of the colors to be filtered should be adjusted if the average brightness in the image is too bright or too dark. This is implemented by using the computed average pixel brightness in the *previous* image. Since images are captured in rapid sequence there is not a major brightness difference between two images. A separate unit is used to *accumulate pixel values*

for the whole image. To avoid a large division unit, the division of the accumulated sum by the number of pixels in the image is undertaken externally. Thus, we have included a connection between the processor and the FPGA in Fig. 4. The division is not a very computational demanding task since it is undertaken only once for each image.

The *RAM control* unit is the interface to the external RAMs and includes a finite state machine to generate appropriate wait states and control signals. We expect that both the input RAM and the output RAM can be controlled by a common address counter. The system is designed to process images of size 640 x 480. That is, each image consists of 307,200 pixels. This results in an address bus with 19 bits. The RAM is expected to have an access time of 8 ns. The RAM on each side of the FPGA would be of a size much larger than the image and separate parts in each of these RAMs can be used for writing and reading, respectively. However, to provide fast access without bus conflicts, two RAMs could be needed at both the input as well as the output side of the FPGA.

The thresholds for each of the three colors (red, white and black) are computed as explained in Section 2. The operation of the multiplexer is given in Table II. This outputs the value the *thresholds* are to be adjusted with. If the image is too light, the value will be *added* to thresholds. If the image is too dark on the other hand, the value is *subtracted* from the thresholds.

TABLE II
THE OPERATION OF THE MULTIPLEXER (MUX) IN FIG. 5.

Input	MUX Ctrl	MUX Output	Comment
I ₀	0 0	0	
I ₁	0 1	125 - Pixel average	Too dark image
I ₂	1 0	Pixel average - 150	Too light image
I ₃	1 1	0	

Each of the thick lines containing one pixel color in the figure consists of 8 bits. All registers/counters in the design is controlled by a synchronous clock signal (not shown). The design is synthesized for Virtex II PRO (XC2VP7-FG456-7). So far, only the reconfigurable logic is in use. However, we plan to continue the work to have the software running on the Power PC processor on the FPGA.

4 Results

This section reports about the experimental work undertaken. The synthesis (place-and-route) resulted in a minimum clock period of 7.5 ns (maximum frequency of 133 MHz). Only 3% of the CLBs in the device were used. Without using interleaved RAM, we need 6 clock cycles for each read/write (undertaken in parallel). This results in 13.8 ms for filtering (Image filtering RWB) one complete image. This is a 72% reduction of time needed for this operation. Adding the software part, the full filtering operation requires now 94 ms. As mentioned in Section 3 however, the system

offers pipelining between the FPGA and the processor. Thus, the processor is now the bottleneck of this pipeline and the time needed for each image is 80 ms. This results in the system being able to process 12.5 images per second rather than only 8.

Only a small part of the logic in the device is used to store the present system and the future work consist of converting other parts on the system into hardware. This will be highly needed since the software will be running much slower (300 MHz) on the embedded processor in the FPGA compared to on a PC. Further, by having the software running on the built-in system, it will provide a System-On-Chip for image recognition.

5 Conclusions

The paper has presented a novel mapping of image filtering into reconfigurable logic as a part of a speed limit detection system. By this approach, we are able to obtain a larger number of images processed per second. By implementing the *most* computational part in hardware, we have completely eliminated the time needed for this part in the proposed pipelined system. The work is focused on reducing computation time at the same time as getting high recognition performance. Thus, the hardware implementation is equivalent to software regarding the quality of the filtering operation.

References

- [1] H. Akatsuka and S. Imai. Road signpost recognition system. In *Proc. of SAE vehicle highway infrastructure: safety computability*, pages 189–196, 1987.
- [2] A. Broggi et al. Design and implementation of the paprica parallel architecture. *Journal of VLSI Signal Processing*, 19(1):5–18, May 1998.
- [3] B.A. Draper et al. Accelerated image processing on FPGAs. *IEEE Transactions on Image Processing*, 12(12):1543 – 1551, Dec 2003.
- [4] E. Jamro and K. Wiatr. Convolution operation implemented in FPGA structures for real-time image processing. In *Proc. of the 2nd International Symposium on Image and Signal Processing and Analysis (ISPA'2001)*, pages 417 – 422, 2001.
- [5] M. Lalonde and Y. Li. Road sign recognition, survey of the state of the art. In *CRIM/IIT (Centre de recherche informatique de Montreal)*, 1995.
- [6] S. McBader and P. Lee. An FPGA implementation of a flexible, parallel image processing architecture suitable for embedded vision systems. In *Proc. of Parallel and Distributed Processing Symposium*, pages 228 – 232, 2003.
- [7] F. Morgan P. McCurry and L. Kilmartin. Xilinx FPGA implementation of an image classifier for object detection applications. In *Proc. of International conference on image processing*, pages 346–349, 2001.
- [8] L. Sekanina. *Evolvable Components, From Theory to Hardware Implementations*. Springer-Verlag, 2004. ISBN 3-540-40377-9.
- [9] L. Sekanina and J. Torresen. Detection of Norwegian speed limit signs. In *Proc. of the 16th European Simulation Multiconference (ESM2002)*, pages 337–340. SCS Europe, June 2002.
- [10] R. Thomas. Less is more [intelligent speed adaptation for road vehicles]. *IEE Review*, 49(5):40–43, May 2003.
- [11] J. Torresen, J.W. Bakke, and L. Sekanina. Efficient recognition of speed limit signs. In *Proc. of 7th IEEE International Conference on Intelligent Transportation Systems (ITSC2004)*, 2004.
- [12] J. Torresen, J.W. Bakke, and L. Sekanina. Recognizing speed limit sign numbers by evolvable hardware. In *Proc. of Parallel Problem Solving from Nature VIII (PPSN VIII)*, Lecture Notes in Computer Science. Springer-Verlag, 2004.