Overhead wires detection by FPGA real-time image processing

Paweł Kowalski[1*] and *Robert* Smyk[1]

[1]Gdansk University of Technology, 80-233 Gdansk, Poland

Abstract. The paper presents design and hardware implementation of real-time image filtering for overhead wires detection divided on image processing and results presentation blocks. The image processing block was separated from the whole implementation, and its delay and hardware complexity was analysed. Also the maximum frequency of image processing of the proposed implementation was estimated.

1 Introduction

Nowadays, electrical energy is widely available and consumed. One of the main transmission mediums are overhead lines. It is essential to minimize the influence of potential failures of such lines by performing frequent inspections. This task can be achieved manually or partly automated using inspection robots like drones [1–4]. In this case, the research is focused on the impact of distance from the line on drones operation and the possibility of landing on wires. Also the prototypes of the systems for aircraft collision avoidance with lines [4] are known. In both cases the precise information about the relative position of the wire is required. In this paper the fast image filtering for wire detection and the analysis of its hardware implementation are shown.

2 Wire detection algorithm

The algorithm consists of two main steps: edge detection and edge reduction. Edge detection procedure is based on using the 3x1 mask in the following form

$$P_{3\times 1} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}. \tag{1}$$

The algorithm provides relatively high sensitivity in strengthening horizontal edges and at the same time is insensitive to vertical edges [1–6]. The advantage of the algorithm is rapid execution and small amount of resources necessary for its implementation in the FPGA [4–5]. It allows to design a filter working with a delay of two pixels. When the pixel on position *y* is received, the filtration result for position *y-2* is known. The formula used for edge detection is in the following form

$$z_{x,y} = v_{x,y-1} - v_{x,y+1}\,, \tag{2}$$

where $v_{x,y}$ is a pixel of image *V* (matrix with grayscale pixels) with coordinates (*x, y*). The result of filtration of the image *V* (2) is the matrix *Z*. The final step of the edge detection is thresholding performed on the matrix *Z*, resulting in the matrix *Z'* (2) containing

$$z'_{x,y} = 1 \quad when \quad \begin{cases} 1 & if \quad z_{x,y} \ge T \\ -1 & if \quad z_{x,y} \le -T \\ 0 & otherwise \end{cases} \tag{3}$$

where $z'_{x,y}$ is an element of the *Z'* matrix with coordinates (*x, y*).

The wire in the image is visible as a line with a slight curvature. Such a line consists of two opposite parallel edges, which in the case of (3) gives the result with the opposite signs. In the edge reduction step the opposite edges are detected. In this step two edges are reduced to one line. At this stage new matrix *W* is created with the same size as the *Z* array but initialized by zeros

$$w_{x,\lfloor y+d/2 \rfloor} = 1\; if \; \begin{cases} \forall h \in [1; d-1] : z'_{x,y+h} = 0 \\ z'_{x,y} = 1 \\ z'_{x,y+d} = -1 \end{cases} ,d{=}1,2,..d_{max} \tag{4}$$

where d_{max} is the maximum thickness of the detecting wire. This value can be selected based on the approximate wire thickness, distance from the wire and image parameters.

3 Hardware Implementation

The digital camera sends an image in a series of pixels. In the standard approach using a PC, processing starts ones the whole frame has been transmitted to memory. In the FPGA environment, the processing can be performed in the same time when the pixel data of frame is being transmitting. In comparison, when the FPGA finishes the processing, the processing in standard PC only starts.

The whole processing path was implemented in the Intel Cyclone V FPGA embedded in Terasic DE10-Nano development board. The raw image data is obtained by a digital camera OV7670.

* Corresponding author: pawel.kowalski2@pg.edu.pl

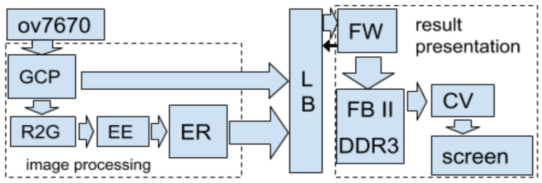

Fig. 1. Block diagram of the hardware implementation.

Fig. 1 shows a block diagram of the implemented processing path. It consists of two main sections: image processing and results presentation. A middle element is the *LineBuffer* module (*LB*), where the incoming data in the form of a raw image from the camera and the final processed image are assembled into one line of the output image.

The lines are assembled, buffered and displayed on the monitor during the result presentation procedure. The combined image consists of a raw image from the camera and a processed image. Buffering is needed due to the frequency difference between the camera and the monitor. The section of the result presentation was built using *Frame Buffer II IP Core* (*FB II*) and *Clocked Video IP Core* (*CV*). These cores are available in the *Altera Quartus 16.1* environment. *FB II* is buffering the image in the DDR3 RAM. The *FrameWriter* (*FW*) module takes subsequent pixels from the *LB* and sends them to the *FB II* and the *CV* transmits the image from *FB II* to the LCD monitor via HDMI.

The image processing procedure begins by collecting raw data from the camera. The camera sends data via 8-bit parallel interface. A single pixel is sent in two 8-bits data packets containing 15 significant bits, 5 bits per each RGB component. Collecting the data from the camera has been carried out in the *GetCamPix* module (*GCP*). It generates three signals at the output:

- pixel value (three RGB components, 5 bits each),
- *x* and *y* coordinates (11 bits each).

These signals are sent simultaneously to the *LB* and *RgbToGray* (*R2G*) modules. *R2G* in real time converts pixels to grayscale using a formula (5)

$$Gray = 3 * Red + 6 * Green + Blue \qquad (5)$$

The gray scale pixels are used in *EdgeExtractor* (*EE*). The *EE* module performs edge extraction (2) on a signal from *R2G* module. The resultant value generated by the *EE* is in the *SM* code (*sign-magnitude*). Further identification of the edge type is determined by the most significant bit (MSB). MSB=0 denotes positive and MSB=1 the negative edge. The result of the *EE* is used in the *ER* module for opposite parallel edges detection. The pixel position is the output from the module after edge reduction. Every obtained non-zero position gives detected line (pixel=1), where other pixels=0.

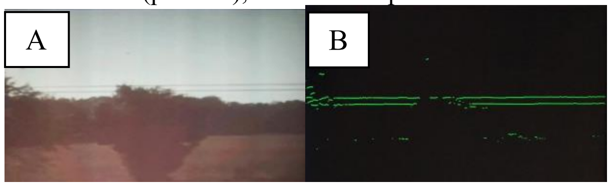

Fig. 6. Example of wire detection in the image, A – raw image from the camera, B – filtered image.

Fig. 6 shows an example of wire detection using the described method. In the experiment, the threshold *T* and the maximum thickness of wire d_{max} were selected experimentally.

4 Hardware analysis

The hardware amount and maximum operational frequency for the implemented structure were determined using *TimeQuest Timing Analyzer* from the *Quartus Prime* package. The main modules: *EdgeExtractor* and *EdgeReductor* were analyzed. Other modules have been omitted. The maximum pipelining frequency was 245 MHz for *EE* and 240 MHZ for *ER*. This corresponds to an image stream with a maximum frequency of 780 frames/s at a resolution of 640x480 or 115 frames/s for a FullHD video (1920x1280).

Tab. 1. Synthesis results in FPGA Cyclone V.

	ALM	block memory bits	registers	F_{max} [Mhz]	max [fps]	
					640x480	FullHD
EE	15	0	5	245	797	118
ER	35	0	74	240	780	115

5 Conclusion

The paper presents the design and hardware implementation of real-time image filtering for overhead wires detection. The proposed procedure is characterized by low latency and gives a small footprint in the FPGA resources. Each significant stage of the algorithm is fully pipelined. The effectiveness of the detection with the proposed solution was tested experimentally in the real environment.

References

1. P. Kowalski, R. Smyk, Pozn Univ Technol Acad J, Electr Eng, **96**, 255–266 (2018)
2. P. Kowalski, M. Czyżak, Pozn Univ Technol Acad J, Electr Eng, **96**, 243–254 (2018)
3. P. Kowalski, M. Czyżak, R. Smyk, ITM Web Conf., 1044 (2018)
4. P. Kowalski, R. Smyk, Pozn Univ Technol Acad J, Electr Eng, **100**, 99–110 (2019)
5. P. Kowalski, R. Smyk, Zeszyty Naukowe Wydziału Elektrotechniki i Automatyki Politechniki Gdańskiej, **61**, 49–52 (2018)
6. J. B. Burns, A. R. Hanson, E. M. Riseman, IEEE Transactions on Pattern Analysis & Machine Intelligence, **8**, 4, 425–455 (1986)