© Journal of the Technical University - Sofia
Plovdiv branch, Bulgaria
"Fundamental Sciences and Applications"
Vol. 25, 2019

Optimal speed algorithm for calculating high and low thresholds to be used in FPGA based Canny focused on maximum performance

Dimitre Kromichev

Department of Marketing and International Economic Relations, University of Plovdiv, 24 Tzar Asen Street, Plovdiv 4000, Bulgaria

dkromichev@yahoo.com

Abstract. The paper presents an optimal speed algorithm for accurately calculating high and low thresholds. It is to be used in FPGA based Canny targeting maximum performance. Scrutinized is the proposed algorithm's computational mechanism. Its functionality is set forth in detail. Proved is the algorithm's practicality in terms of guaranteeing the calculation of the most appropriate threshold values in a dynamic fashion. On a comparative basis, explored amd analyzed are the proposed algorithm's capabilities to ensure reliable results at optimal speed under all test conditions.

1. Introduction

Speed has two capital parameters: the highest clock frequency of execution and the minimum number of clock cycles needed to secure an exact result at the highest clock rate. Selecting the appropriate high and low thresholds is crucial to the detected contours' precision.

The objective of this paper is to propose an optimal speed algorithm for calculating high and low thresholds to be used in FPGA based Canny targeting maximum performance on the platform of total mathematical accuracy. The task is to scrutinize the presented algorithm in terms of its computational mechanism and explore its performance with respect to the two capital speed parameters. Relevant to the conducted experiments and conclusions arrived at are only gray-scale images. The software tool utilized to to ascertain the total number of high and low threshold selection options is Scilab. The targeted hardware is Intel (Altera) FPGAs. The following ten Intel (Altera) FPGA families are used in the tests: 130 nm, 90 nm, 65 nm, 40 nm, 28nm, Cyclone, Cyclone II, III, IV, V; 130 nm, 90 nm, 65 nm, 40 nm and 28nm Stratix, Stratix II, III, IV, V. Intel (Altera) Quartus, ModelSim and TimeQuest Timing Analyzer are utilized for exploring the feasibility and practicality of the proposed algorithm.

2. Survey of the available approaches

In distributed Canny on Xilinx Virtex 4 and 5 [1[[2][6][7][10], the thresholds' calculation is modified to enable parallel processing. A 64-bin uniform discrete histogram is used for the high threshold calculation. This entails performing 64 multiplications and comparisons. All simulations are for 512x512 images and are performed at a maximum clock rate of 100 MHz. It is pointed out [3][4] that Canny is hard to work in real time due to its computational complexity. The proposed implementation uses block classification for adaptive thresholding. Direct implementation of Canny algorithm is too slow to be employed in real-time applications [8][9][12]. To enable parallel processing the input image is divided into m×m overlapping blocks. Uniformly quantized gradient magnitude histograms are computed on overlapped blocks. A threshold selection algorithm based on the distribution of pixel gradients in a block of pixels is used. The high threshold is computed on the basis of the histogram. The lower threshold is 40% of the higher threshold. Setting the thresholds by experience requires repeated tests to find the appropriate value [11]. It is underlined that needed is an adaptive method to automatically select the most appropriate thresholds by combining the image characteristics [5].

With respect to the two capital speed parameters, none of the aforesaid approaches are up to accomplishing the goal of achieving maximum performance in FPGA based Canny.

3. The proposed algorithm for calculating high and low thresholds

The task to start with is: define a staple set of minimum steps between two successive high threshold values and select sets of ratios between high and low thresholds, and on that basis, ascertain the possible number of thresholds with respect by steps and ratios. The achieved results are:

Table 1. Total number of high and low threshold selection options.

Parameters	Values					
	Minimum step between two successive high threshold values					
	5		7		10	
	Ratios b/n high and low threshols		Ratios b/n high and low threshols		Ratios b/n high and low threshols	
	2, 2.2, 2.5, 2.8, 3	2, 2.5, 3	2, 2.2, 2.5, 2.8, 3	2, 2.5, 3	2, 2.2, 2.5, 2.8, 3	2, 2.5, 3
High thresholds	50	50	36	36	25	25
High threshold range	[5,250]	[5,250]	[7,252]	[7,252]	[10,250]	[10,250]
Pairs of thresholds	250	150	180	108	125	75

The goal of the proposed algorithm is to provide optimally fast computing of the high and low threshold values by satisfying the following computational requirements: 1) Compute the thresholds in a dynamic fashion - simultaneously with the non-maximum suppression, to boost pipelining; 2) Total mathematical accuracy; 3) Utilize only the fastest FPGA arithmetic; 4) Avoid iterative calculations.

The algorithm encompasses the following sequence of steps:

1) Define the range of positive integers the high threshold calculation will be based upon. It represents a closed interval whose minimum and maximum values *Nmin* and *Nmax* are selected such that the result of *[(Nmax+1) – Nmin]* is equal to a number whose factor is 7, and *(Nmax+Nmin)* $< 2^8 - 1$.

2) The interval from *step 1)* is divided into equal subintervals whose minimum and maximum values *SUB_Nmin* and *SUB_Nmax* are related by the equation: *[(SUB_Nmax+1) – SUN_Nmin]* = 7. Each subinterval is associated with one reference value which is equal to the fourth out of the seven consecutive values contained in this subinterval.

3) Define a set of table values – they are positive integers calculated by dividing all the values in the interval [5, 250] by each of these five values {2, 2.2, 2.5, 2.8, 3} prior to starting executing the algorithm.

4) Define a set of counters. Each counter is associated with one of the subintervals from *step 2)*.

5) Starting with non-maximum suppressed image pixel #1, each pixel is assessed with respect to its value falling or not falling within one of the subintervals from *step 2)*. If the pixel value is within an interval, the counter associated with this particular interval is incremented.

6) With all the pixels having been subjected to the assessment procedure from *step 5)*, the values in each of the counters defined in *step 4)* are compared to select the two largest among them. On the basis of these largest values, one of the subintervals and the reference value associated with it is compared with the other of the two largest intervals and its reference value.

7) Calculate the difference between the two reference values for the purpose of further defining a new interval. The number of pixels in this new interval is equal to the difference between the two reference values. In the interval thus defined, the leftmost value is equal to the larger of the two selected reference values in *step 6)*. The rightmost value is equal to the smaller of the two selected reference values in *step 6)*.

8) Explore the values within the new interval defined in *step 7)* to ascertain the smallest among them. This value is the high threshold.
9) The high threshold is associated with a set to five table values calculated in advance by using divisors {2, 2.2, 2.5, 2.8, 3} as described in *step 3)*.
10) Select the appropriate table value from *step 3)*. This is the low threshold.
The compitational mechanism of the algorithm is presented in Figure 1.

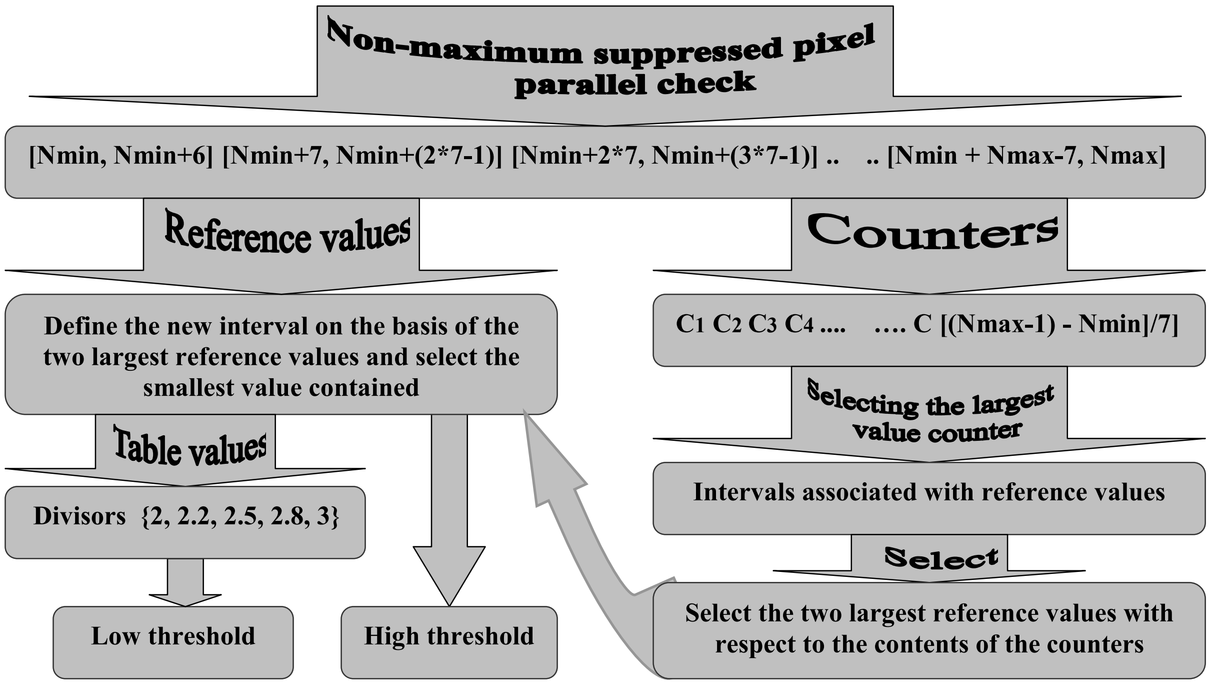
Non-maximum suppressed pixel parallel check
[Nmin, Nmin+6] [Nmin+7, Nmin+(2*7-1)] [Nmin+2*7, Nmin+(3*7-1)] .. .. [Nmin + Nmax-7, Nmax]
Reference values
Counters
Define the new interval on the basis of the two largest reference values and select the smallest value contained
C1 C2 C3 C4 .... C [(Nmax-1) - Nmin]/7]
Selecting the largest value counter
Table values
Divisors {2, 2.2, 2.5, 2.8, 3}
Intervals associated with reference values
Select
Low threshold
High threshold
Select the two largest reference values with respect to the contents of the counters

Figure 1. Functional model of the algorithm for calculating high and low thresholds

4. Exploring the algorithm's mathematical accuracy

The table values are calculated by using the values within the interval [5, 250] as dividends, and the divisors are each of {2, 2.2, 2.5, 2.8, 3}. They are computed in advance for the purpose of avoiding delays in the very essential process of calculating the thresholds. The table values are integers stored prior to starting Canny's execution, thus no mathematical operation is required to be used. In view of the fact that there are five divisors, there is an option to switch between the different tables containing results obtained by utilizing different divisors. This renders the selection of the low threshold more adaptive and flexible, depending on the particular input image statistics. The aim is to ensure an optimal range of options without resorting to additional integer arithmatic calculations.

5. Exploring the algorithm's speed capabilities in FPGA

To evaluate the proposed algorithm's performance, employed is the following methodology: 1) Select ten real life images of equal size and execute Canny using Scilab Image and Video Processing Toolbox, Scilab Image processing Design Toolbox; 2) Store the ten image matrices obtained after executing the non-maximum suppresssion module into separate files using Scilab; 3) Define ROM using Quartus; 4) Open a Memory initialization file (.mif) in Quartus; 5) Store each non-maximum suppressed image matrix into a .mif file; 6) Write VHDL programs to implement the algorithms being compared; 7) Comparatively evaluate the proposed algorirthm's performance using non-uniform quantizing and in-class variance algorithms in terms of the two capital speed paremeters - the highest clock frequency and the minimum number of clock cycles needed to secure an exact result at the highest clock rate, by conducting performance tests on the ten non-maximum suppressed image matrices.

The achieved results are exhibited in the tables below.

Table 2. Capital speed parameters for the high and low threshold calculation using the proposed algorithm.

FPGA family	Values										
	Capital speed parameters for the high and low threshold calculation using the proposed algorithm										
	Maximum clock frequency (MHz)	Total number of clock cycles taken to secure mathematically accurate result									
		Non-maximum suppressed image matrices									
		1	2	3	4	5	6	7	8	9	10
Cyclone	196	68	75	80	73	81	67	88	76	82	89
Cyclone II	246	68	75	80	73	81	67	88	76	82	89
Cyclone III	349	68	75	80	73	81	67	88	76	82	89
Cyclone IV	367	68	75	80	73	81	67	88	76	82	89
Cyclone V	385	68	75	80	73	81	67	88	76	82	89
Stratix	291	68	75	80	73	81	67	88	76	82	89
Stratix II	385	68	75	80	73	81	67	88	76	82	89
Stratix III	502	68	75	80	73	81	67	88	76	82	89
Stratix IV	548	68	75	80	73	81	67	88	76	82	89
Stratix V	586	68	75	80	73	81	67	88	76	82	89

Table 3. Capital speed parameters for the high and low threshold calculation using the non-uniform quantizing algorithm.

FPGA family	Values										
	Capital speed parameters for the high and low threshold calculation using the non-uniform quantizing algorithm										
	Maximum clock frequency (MHz)	Total number of clock cycles taken to secure mathematically accurate result									
		Non-maximum suppressed image matrices									
		1	2	3	4	5	6	7	8	9	10
Cyclone	149	885	946	962	935	892	928	912	964	899	908
Cyclone II	192	885	946	962	935	892	928	912	964	899	908
Cyclone III	228	885	946	962	935	892	928	912	964	899	908
Cyclone IV	231	885	946	962	935	892	928	912	964	899	908
Cyclone V	234	885	946	962	935	892	928	912	964	899	908
Stratix	226	885	946	962	935	892	928	912	964	899	908
Stratix II	302	885	946	962	935	892	928	912	964	899	908
Stratix III	426	885	946	962	935	892	928	912	964	899	908
Stratix IV	442	885	946	962	935	892	928	912	964	899	908
Stratix V	464	885	946	962	935	892	928	912	964	899	908

Table 4. Capital speed parameters for the high and low threshold calculation using the in-class variance algorithm.

FPGA family	Values										
	Capital speed parameters for the high and low threshold calculation using the in-class variance algorithm										
	Maximum clock frequency (MHz)	Total number of clock cycles taken to secure mathematically accurate result									
		Non-maximum suppressed image matrices									
		1	2	3	4	5	6	7	8	9	10
Cyclone	149	1884	1928	1846	1952	1908	1943	1959	2008	1949	1918
Cyclone II	192	1884	1928	1846	1952	1908	1943	1959	2008	1949	1918
Cyclone III	228	1884	1928	1846	1952	1908	1943	1959	2008	1949	1918
Cyclone IV	231	1884	1928	1846	1952	1908	1943	1959	2008	1949	1918
Cyclone V	234	1884	1928	1846	1952	1908	1943	1959	2008	1949	1918
Stratix	226	1884	1928	1846	1952	1908	1943	1959	2008	1949	1918
Stratix II	302	1884	1928	1846	1952	1908	1943	1959	2008	1949	1918
Stratix III	426	1884	1928	1846	1952	1908	1943	1959	2008	1949	1918
Stratix IV	442	1884	1928	1846	1952	1908	1943	1959	2008	1949	1918
Stratix V	464	1884	1928	1846	1952	1908	1943	1959	2008	1949	1918

Table 5 presents the resources utilized by the proposed algorithm in Intel (Altera) Cyclone V E 5CEBA4F17C6N Device**.** The exibited data proves that the algorithm's advanced computational mechanism ensures its being quite economical in utilizing the FPGA resources. This is mainly due to the optimized number of essential arithmetic operations and the technology of handling the entire image statistics.

Table 5. Resources utilized by the proposed algorithm in Intel (Altera) Cyclone V E 5CEBA4F17C6N Device.

FPGA Resource	Resource Counts	Used	Resource utilization
Logic utilization (in ALMs)	18480	228	1.2337 %
Total registers	-	80	-
Total block memory bits	3168512	0	0 %
Total DSP blocks	66	0	0 %

6. Analysis of results

The data in Table 2 shows that the proposed algorithm is capable of executing at optimal clock frequency in each of the targeted FPGA platforms. This is due to the fact that all comparison functions are replaced with subtraction and checking the result for being larger or equal to zero.

The proposed algorithm demonstrates two substantial advantages: 1) It calculates the exact high threshold value on the basis of using intervals at two levels; 2) All the clock cycles required for calculating the leftmost and the rightmost values of the lower level interval are equal to *the number of counters + 1.* The lower level interval having been defined, its values are compared on the basis of the same subtraction procedure as the higher level intervals. The number of values to be compared here varies and depends on the difference between the leftmost and the rightmost values. It is this fact that determines the difference between the number of clock cycles required to calculate the two thresholds for the tested ten non-maximum suppressed image matrices (Table 2).

The proposed algorithm selects the low threshold value directly from a table of values computed in advance. Thus, the high threshold having been calculated, there is no need of any further arithmetic operations to obtain the low threshold. Therefore, in terms of clock cycles required to compute the two thresholds, the proposed algorithm is most favourable to optimal pipelining efficiency.

Both the non-uniform quantizing and in-class variance algorithms rely on comparison function to determine the largest or the smallest values. Hence, their execution frequencies are lower. In terms of clock rate, another very unfavourable aspect is that they use division to calculate the low threshold. The maximum clock frequencies exhibited in Tables 3 and 4 represent non-uniform quantizing and in-class variance implementations in which division is replaced with calculating the low thresholds by using precomputed reference table values. In this way, the comparison between the proposed algorithm and the non-uniform quantizing and in-class variance algorithms is drawn on equal terms. If conventional division had been used instead, the naximum frequencies of the non-uniform quantizing and in-class variance algorithms would have been from 5.2 to 6.4 times as low. Their computational mechanisms being entirely based on iterations, the total number of clock cycles taken to calculate the two thresholds is from 12.7 to 32.5 times larger than the number of clock cycles required by the proposed algorithm (Tables 2, 3 and 4).

For the tests, it is assumed that all the algorithms start the computation of the high threshold under the same conditions – the data regarding the non-maximum suppressed image statistics has already been obtained in compliance with each of the targeted algorithms' specifics. And here is the crucial difference between the proposed algorithm and the other two approaches. The proposed algorithm acquires the necessary information on the image statistics dynamically – simultaneously with the non-maximum suppression execution. Therefore, as soon as all pixels have been processed in the non-maximum suppression module, the calculation of the high threshold can start without any delay. The non-uniform quantizing and in-class variance algorithms acquire the necessary information in sequential fashion – after the non-maximum suppression computations have finished. Therefore, delays equal to the number of clock cycles taken to perform operations on the entire non-maximum suppression image matrix must be added to the clock cycles exhibited in Table 3. This results in blocking the pipelining for additional number of clock cycles proportional to the image size.

Considering both capital speed parameters, the proposed algorithm is from 19.9 to 48.7 times faster than the non-uniform quantizing and in-class variance. It is optimal in the speed domain.

7. Conclusion

Proposed is an optimal speed algorithm for calculating high and low thresholds in FPGA based Canny. The algorithm's computational mechanism is presented in detail. On a comparative basis, scrutinized are the proposed algorithm's speed capabilities in FPGA, For the purpose, developed is a specific methodology to test speed with respect to the two capital paremeters: the highest clock frequency of execution and the minimum number of clock cycles needed to secure an exact result at the highest clock rate. Test results show that the proposed algorithm is from 19.9 to 48.7 times faster than the two most widely used approaches for calculating high and low thresholds in FPGA based Canny - non-uniform quantizing and in-class variance. The analysis of experimental results proves that it is most favourable to pipelining efficiency. Therefore, the proposed algorithm for dynamically calculating high and low thresholds can serve as a reliable building block in FPGA based Canny focused on maximum performance.

References

[1] Aasiya Anjum and Sanjay Asutkar 2015 FPGA Implementation of Efficient Edge Detection Using Canny Algorithm *International Journal on Recent and Innovation Trends in Computing and Communication* Vol. **3** (2), pp. 20-22

[2] Divya. D, Sushmap S. 2013 FPGA Implementation of a Distributed Canny Edge De*tector International Journal of Advanced Computational Engineering and Networking* Vol. **1** (5), pp. 46-51

[3] Sangeetha D., Deepa P. 2016 An Efficient Hardware Implementation of Canny Edge Detection Algorithm *2016 International Conference on VLSI Design and 2016 International Conference on Embedded Systems (VLSID)* pp. 817-822

[4] Sangeetha D., Deepa P. 2016 FPGA implementation of cost-effective robust Canny edge detection algorithm *Journal of Real-Time Image Processing (JRTIP)* Vol. **12** (1), pp. 1-14

[5] Fangxin Peng, Xiaofeng Lu, Hengli Lu, Sumin Shen 2012 An Improved high-speed canny edge detection algorithm and its implementation on FPGA *Proceedings of SPIE, Fourth International Conference on Machine Vision (ICMV 2011): Computer Vision and Image Analysis; Pattern Recognition and Basic Technologies, Conference* Vol. **8350**, pp. 402-406

[6] Heena S. Shaikh, Narayan V. Marathe 2017 Implementation Of Distributed Canny Edge Detection Technique *International Research Journal of Engineering and Technology (IRJET)* Vol. **04** (05), pp. 2926-2928

[7] Lakshmamma K M., Chandana B.R 2015 RTL Design and FPGA Implementation of Canny Edge Detector with Real Time Threshold Adjustment Capability *International Journal of Science and Research (IJSR)* Vol. **4** (4), pp. 1731-1733

[8] Poonam S. Deokar and Anagha P. Khedkar 2015 Implementation of Modified Distributed Canny Edge Detector Algorithm Using FPGA *International Journal of Information Research and Review* Vol. **2** (8), pp. 999-1003

[9] Qian Xu, Srenivas Varadarajan, Chaitali Chakrabarti, Lina J. Karam 2014 A Distributed Canny Edge Detector: Algorithm and FPGA Implementation IEEE *Transactions on Image Processing* Vol. **23** (7), pp. 2944 – 2960

[10] Rupalatha T., Leelamohan C., Sreelakshmi M. *2013* Implementation of distributed Canny edge detection on FPGA *International Journal of Innovative Research in Science, Engineering and Technology* Vol. **2**, (7), pp. 2618-2626

[11] Thombare Ashwini, S. B. Bagal 2015 A Distributed Canny Edge Detector with Threshold Segmentation *International Journal of Modern Trends in Engineering and Research IJMTER*, pp. 1567-1572

[12] Veeranagoudapatil, Chitra Prabhu 2015 Distributed Canny Edge Detector: Algorithm & FPGA Implementation *International Journal for Research in Applied Science & Engineering Technology (IJRASET)* Vol. **3** (5), pp. 586-588