Paweł KOWALSKI[1], Robert SMYK[1]

Gdańsk University of Technology (1)

doi:10.15199/48.2023.03.36

Improved Hardware Hough Transform implementation

Abstract. *The paper presents hardware FPGA implementation of the Hough Transform algorithm for digital real time image processing. In the developed hardware structure, the processing efficiency was increased through the use of controlled pipelining, trigonometric arithmetic by look-up, integer operands only and dispersing the voting memory. The presented structure was used experimentally in the real time image processing system implemented as single chip in Intel Cyclone V FPGA. We obtained a constant image processing speed 275 MHz not related to the angle resolution.*

Streszczenie. *W artykule zaprezentowano autorski system przetwarzania obrazu wykorzystuj ˛acy algorytm transformacji Hough. Algorytm został zaimplementowany w pojedynczym układzie FPGA Intel Cyclone V wraz z pełnym torem akwizycji danych z kamery oraz strumieniowania przetworzonych danych do standardu HDMI. Przedstawiona implementacja została zoptymalizowana z uwzgl˛ednieniem specyfiki FPGA, poprzez m.in. wprowadzenie potokowo´sci na poziomie pojedynczego piksela, tablicowanie funkcji trygonometrycznych oraz rozproszenie pami˛eci u˙zytej do realizacji procesu głosowania. W całym torze przetwarzania u˙zyto operandów całkowitych. Uzyskano stał ˛a pr˛edko´s´c przetwarzania 275 MHz niezale˙znie od rozdzielczo´sci k ˛ata.* **(**Ulepszona implementacja sprzętowej transformacji Hougha **)**

Keywords: image processing, FPGA, Hough transform, line detection
Słowa kluczowe: przetwarzanie obrazu, FPGA, transformacja Hougha, wykrywanie linii

Introduction

Dedicated hardware image processing system (IPS) is now a typical component of modern devices equipped with a camera. While such systems can be constructed in many different ways, there is a common pattern in image processing that consists of several steps. In a standard way the input is a camera module, where the image is initially processed at the pixel level, which is a natural effect resulting from the method of byte-by-byte reading data from the image sensor. In further stages, after composing the full frame from the image, extraction of an objects and scene analysis is performed. Finally, the image can be saved with a set of its features or streamed to LCD display. Therefore, the design of a dedicated ISP and the algorithms implemented in it should consider the standard input pipe, processing method and the output pipe. This paper focuses on the implementation of IPS dedicated to line recognition and detection, which is used in image processing in vehicles and aircrafts, also in computer vision, medical image processing and artificial intelligence [1]. One of the known line detection algorithm is the Hough transform, originally described in 1962 [2]. Its special feature is high tolerance to interference and lack of line continuity [3]. One of the disadvantages of the method is its high computational cost due to the need of the multiple memory access. The ability to use it for real time image processing by the software implementations using general purpose processors (CPU) is very limited here. In this paper we show that efficient hardware implementations with parallelized processing stages and proper pipelining are possible. Additionally, the effectiveness can be improved by introducing modifications such as the use of a look-up tables, the CORDIC algorithm and designing a dedicated memory with simultaneous access to many cells [1]. In the following sections of the paper, we present an overview of the methods of implementation of processing using Hough transform, the details of hardware realization of main processing block and finally the experimental results.

Overview of Hough Transform calculation methods

A single line in a two-dimensional space is defined by a vector ψ orthogonal to a line and connected to the origin of the frame of reference. The vector is defined by two factors: ρ and α, where ρ is the length of the vector and α is the angle between the x axis and the vector ψ. The ρ coefficient is determined using the formula (1)

$$\rho = sin\alpha \cdot y + cos\alpha \cdot x, \tag{1}$$

where x and y are the pixel coordinates in the x and y axes respectively. The coefficients α and ρ are used to create a Hough space composed of $n \times d$ cells, where n defines the cardinality of the set of coefficients α, and d the maximum length ρ. The transformation to Hough space is performed on the binary image. For each set pixel (value=1) of this image, cells (α_i, ρ_i) are incremented in an array representing the Hough space. The set of angles $\alpha_1 \dots \alpha_n$ is selected arbitrarily. On the other hand, the coefficients $\rho_1 \dots \rho_n$ are determined using the formula (1). This procedure is called a voting because the series of increments produces an array where each element represents a single line and its value corresponds to the number of pixels on that line.

A review of the Hough transform implementation methods is presented in [1, 4]. The calculation complexity is largely influenced by the procedure of determining votes with the use of trigonometric functions. One way to reduce this is to store the trigonometric values in the array. During hardware image processing in the chip, they will no longer be computed, but only read by a look-up. A slight disadvantage of this approach is the use of additional FPGA resources when the lookup table is implemented using logic function resources rather than as local memory blocks. Another way to increase the efficiency is to use the CORDIC (COordinate Rotation DIgital Computer) algorithm to quickly determine the values of trigonometric functions. This method was used in the implementation of the Hough transform described in [5]. The main advantage of CORDIC is the ability to avoid multiplication by replacing it with binary shifts. An overview of the architectures for its implementation is presented in [6].

The Hough transformation requires the determination of the coefficient ρ as in (1) for many angles α. It should be noted that its slope can be approximated at the stage of edge extraction based on the gradient. This allows to reduce the number of calculated angles. This method was applied in [8]. Another way to reduce the computational cost is to process only a subset of the pixels. This is done by randomly selecting the pixels to be transformed. Examples of such methods are Probabilistic Hougha Transform (PHT), Randomized Hough Transform (RHT) and Monte Carlo Hough Transform (MCHT). They have a lower computational cost than the methods that process all pixels, but they are also more susceptible to distortions and noises in the image [1]. The implementation of

Hough's voting space is also important. Due to the required large amount of space for storing voting results, the external memory can be used [8, 9]. This approach saves FPGA resources, which allows the use of less expensive FPGAs with fewer logical cells. However, external storage bandwidth may limit the maximum resolution and image processing speed. Due to the advances in FPGA synthesis processing, it is now possible to allow a more algorithmic approach to circuit implementation. Currently, it is more profitable to implement an algorithm that takes up more FPGA resources but has higher performance. For example, it is possible to use multiple independent memory blocks within the chip that are accessed in parallel. This approach was used in [8, 10]. In [8], the voting procedure has been divided into modules. Each of them is equipped with double buffered memory. Calculations are made on fixed-point numbers and the angle α is sampled every $11.25°$. It was implemented in FPGA Virtex 4 and allows image processing with a resolution of 800×600 at a speed of 30 fps. In [10] an implementation running on the DE4320C2 board containing the Altera Stratix-IV EP4SGX230KF40C2 FPGA chip is presented. The hardware implementation of the algorithm was used to detect road lanes. It enables the processing of an XGA image frame in 5.4 ms. The implementation is composed of a set of computational modules. Each of them contains a one-dimensional array which is a fragment of Hough space providing parallel access to many memory cells during voting. Each of the parallel modules additionally contains two multipliers and two look-up tables implemented as RAM. Edges are detected using the Sobel operator and the origin of the coordinate system is at the geometric center of the image. To prevent negative ρ, angles α are from the range 0° to 360°. In [11] a system for the detection of road lanes using the Sobel edge operator and the Hough transformation is presented. The hardware implementation was described in C and synthesized using Xilinx Vivado High Level Synthesis (HLS). For the image with a resolution of 720×1280, the speed was 59.13 fps. In [12] the standard procedure of determining votes was modified. The parameters ρ and α defining the straight line are determined by shifting the parallel line to it. For comparison, our hardware implementation of the algorithm allows to process images with a frequency of up to 275 MHz and constant efficiency. This allows for 895 fps real time processing of 640×480 px images.

Hardware implementation of real time image processing

In these section we would like to describe our own hardware implementation of the whole IPS that uses the Hough transform algorithm in real time line detection from digital video stream. In Fig. 1 we have presented a schematic diagram of our system. The main processing module HT consisting hardware implementation of the Hough Transform. For experimental purposes, we combined this module with an OV7670 camera with a resolution of 640×480 px. The multiplexed output signal is fed to HDMI, which enables the result of processing to be displayed on the monitor screen. The entire processing has been implemented as a SoC. We uses a Terasic DE10-nano development board equipped with an Intel Cyclone V 5CSEBA6U23I7 FPGA.

The developed image processing system consists of blocks of image acquisition from the camera (CAM), edge extraction (EE) using the Previtt operator [13], Hough transformation (HT), line drawing (DL) based on the transformation results, overlaying a binary image on a color image (ImgMX) and displaying images on the screen via HDMI. All modules,

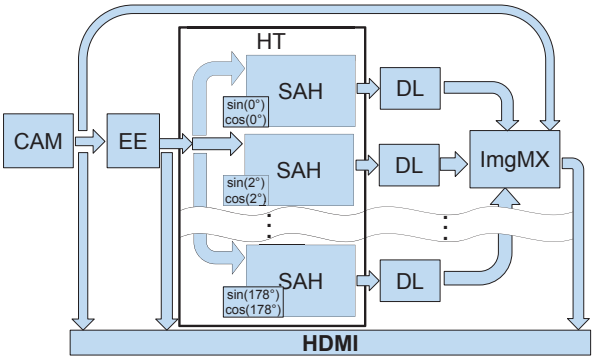

Fig. 1. The scheme of the processing path using the HT module

except for HDMI, are clocked with the clock signal generated in CAM, where the rising edge triggers the loading of a new pixel. The HT block accepts as input a binary image transmitted using the $posX$, $posY$ and $edgeBin$ signals. Pixels marked with coordinates $posX \in < 1; 640 >$ and $posY \in < 1; 480 >$ are pixels of a captured image. The video stream from the camera also contains vertical (pixels marked with $posY = 0$) and horizontal (pixels marked with $posX = 0$) sync signals.

An HT block is made up of a set of blocks *SingleAngleHough* (SAH). Each of them receives sin values determined according to the formula $\lfloor sin\alpha \cdot 128 \rfloor$ and cos determined by $\lfloor cos\alpha \cdot 128 \rfloor$. These signals are constant, determined for each SAH block before the FPGA synthesis. At the same time, the binary image obtained from EE is transmitted to each SAH block via the signals $posX$, $posY$ and $edgeBin$. A single SAH module performs the Hough transform for an angle α and α+180°. In this procedure one row of the array representing the Hough space is filled. For each edge pixel ($edgeBin = 1$) of the input image, an increment of the corresponding memory location is performed, where the cell address defines an integer part of ρ (1). The source code of SAH module in Verilog is presented in Listing 1.

Listing 1. The source code of SingleAngleHough (SAH) module

```
module SingleAngleHough(
 input clk,
 input [9:0] posX,
 input [8:0] posY,
 input edgeBin,
 output reg [37:0] votes,
 output reg cls,
 input signed [8:0] sin,
 input signed [8:0] cos
);

wire clrMem;
wire signed [9:0] centerX;
wire signed [9:0] centerY;
reg signed [18:0] centerP;
reg wren;
reg [9:0] data;
wire [9:0] q;
reg [10:0] clrAddr;
wire [9:0] rdAddr;
reg [9:0] wrAddr;
reg [32:0] edgeBuff;
wire enVote;

assign enVote = (posX==0)? 1'b0 : 1'b1;
assign clrMem = (posY==0)? 1'b1 : 1'b0;
assign rdAddr = clrMem? clrAddr[9:0]:centerP[16:7];
assign centerX = posX-320;
assign centerY = posY-240;

always @(posedge clk) begin
 centerP <= sin*centerY + cos*centerX;
 edgeBuff<={edgeBuff[21:0], rdAddr, edgeBin&enVote};
 if(clrMem) begin
```

```
35    data<=0;
36    cls <= ~clrAddr[10];
37    if (cls) begin
38     clrAddr <= clrAddr+1;
39     wrAddr <= rdAddr;
40     votes<={sin, cos, edgeBuff[10:1], q};
41    end
42   end
43   else begin
44    data = q + edgeBuff[22];
45    wren <= edgeBuff[22];
46    wrAddr<=edgeBuff[21:12];
47    clrAddr<=11'd0;
48   end
49  end
50
51  RAM10io1024 RAM10io1024_HoughWorkSpace
52  (
53   .clock(clk) , // input  clock_sig
54   .data(data) , // input [9:0] data_sig
55   .rdaddress(rdAddr) , // input [9:0] rdaddress_sig
56   .wraddress(wrAddr) , // input [9:0] wraddress_sig
57   .wren(wren | cls) , // input  wren_sig
58   .q(q)  // output [9:0] q_sig
59  );
60  endmodule
```

There are three basic sections in the SAH module: asynchronous section (lines 25–29), synchronous section (lines 31–49) and RAM (lines 51–59). In the asynchronous section the flags $enVote$, $clrMem$ are set/unset and the signals $rdAddr$, $centerX$ and $centerY$ are determined. The flag $clrMem$ determines the operating mode: voting ($clrMem = 0$) or memory clearing with simultaneous results transferring ($clrMem = 1$). The flag $enVote$ is responsible for enabling the voting. The signal $rdAddr$ contains address for reading from memory. The source of this signal is changed depending on the operating mode specified by the flag $clrMem$. The signals $centerX$ and $centerY$ represent the pixel position normalized to the geometric center of the image $(320, 240)$. Normalization increases the efficiency of memory use. Before normalizing ρ takes values in the range of $(0; 800)$, and α is from the range of $(-90°; 180°)$. After normalization ρ is from the range $(-400; 400)$, and α is from the range $(0°; 180°)$. In the first case, the size of the voting array is 270×800. In the latter case, however, a smaller array of size 180×800 is required. The synchronous section is triggered by the rising edge of clk with each new pixel that appears. The $centerP$ and $edgeBuff$ values are determined each cycle. In the $centerP$ register a coefficient ρ (1) is stored. The values of sin and cos are scaled up, which causes that the value of this coefficient is also scaled up by 128. In this case 10 most significant bits represents integer part and 8 least significant bits represents the factorial part of the number. The integer part (10 bits) is used for memory addressing. The shift register $edgeBuff$ stores the address $rdAddress$ and edge $edgeBin$ from last three clock cycles. In the synchronous section, there are also voting and result transmission blocks with memory clearing. The block for transmitting results with clearing the memory (lines 34–42) is performed after the voting is finished, signaled by the set state of the $clrMem$ flag. In this block, successive votes readed from memory are stored in the $votes$ register, which is an output port. At the same time, the read cells are cleared by assigning the value 0. During the procedure of sending votes and clearing memory, the cls flag is also set.

In SAH we used dual port RAM (RAM10io1024) organized as 1024×10 bits. It is used to vote for straight lines by ρ and α coefficients. The coefficient α is obtained in HT and have a constant value. However, the coefficient ρ is determined using (1). For the processed image with a resolution of 640×480, it takes values from -400 to 400. So using 1024 memory cells causes 224 cells to be left unused. It should be noted that in the FPGA system the memories are built of M10K blocks with a capacity of 10Kb, so one block will be used up both in the case of creating memories with a capacity of 800 and 1024 cells. The capacities of 1024 cells are selected because it allows address determination without a sum or difference operation, but only by selecting 10 bits of $centerP$ register.

Experimental Results

Our solution in the form of the hardware ISP (Fig. 1) was applied to image processing in real time. The processing path includes an HT block consisting of 90 SAH modules, allowing for line detection with $2°$ sampling resolution. An example of results is shown in Figure 2.

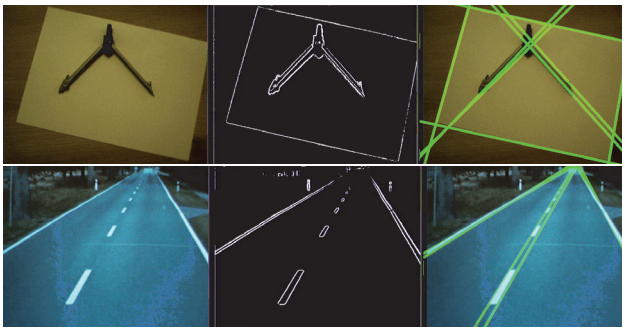

Fig. 2. An example of the results of image processing using the presented method

There are two examples of image processing using the presented method. Each of them consists of three images: a raw image, a binary image with detected edges and a raw image with superimposed detected lines. The developed structure was analyzed. The speed and the amount of resources occupied by the HT module in the version with sampling $\Delta\alpha$ equal to $1°$, $2°$, $5°$ and $15°$ were tested. Changing the sampling angle α is possible by changing the number of SAH blocks used. The developed implementation was compared with the implementations described in [1, 2, 3, 7, 8, 14]. The results of the comparison are collected in Table 1. We compared the resolution of the image used for testing the method, the maximum speed of processing this image specified in frames per second fps, maximum speed of processing subsequent pixels f_{max}, the number of required programmable logic elements CLB (Configurable Logic Blocks), the number of registers reg, the amount of RAM, and sampling the angle of the detected lines $\Delta\alpha$. It should be noted that the presented implementations were realized with the use of other custom hardware solutions and synthesized in various FPGA environments, which may impact on the obtained processing speeds and the amount of used hardware resources.

In this work the HT modules with sampling rates $\Delta\alpha$ equal to $1°$, $2°$, $5°$ and $15°$ were benchmarked. In Table 1 it is marked as HT$\Delta\alpha$. Increasing the accuracy of the slope angle detection requires more resources, but it should be noted that the maximum processing frequency determined by Quartus Time Analysis is constant 275 MHz.

The smallest number of CLBs is in the implementation proposed in [9], but it allows processing at a maximum speed of 27.1 Mhz. The lowest number of registers is achieved by [10] at the pixel processing speed of 147 MHz. The highest processing speed was presented in [14] and in our solution. The both structures runs at a maximum speed of 275 MHz, but in [14], the processing speed drops to 258 MHz

Table 1. Comparison of hardware implementations of the Hough transform.

method	resolution	fps	f_{max} [Mhz]	CLB	reg	RAM [b]	$\Delta\alpha$
HT1	640×480	895	275	4766	18720	1843200	1°
HT2	640×480	895	275	2383	9360	921600	2°
HT5	640×480	895	275	952	3744	368640	5°
HT15	640×480	895	275	319	1248	122880	15°
[14]	640×480	833	258	6301	18900	1843200	1°
[14]	640×480	895	275	421	1260	122880	15°
[8]	600×800	30	14.4	N/A	N/A	N/A	11.25°
[9]	256×256	413	27.1	324	N/A	N/A	1°
[10]	1024×768	185	147	850	645	1513721	2°
[11]	1280×720	60	54.37	77658	66186	8476	9°
[12]	640×480	736	226.76	5717	6010	936	1°

as the $\Delta\alpha$ decreases. Our implementation presented in this paper have a constant efficiency for $\Delta\alpha \in < 1^\circ; 15^\circ >$. Moreover we reduced the number of CLBs by 25% and and the number of registers by 1%, compared to the method previously published [14].

Conclusion

The article presents improved hardware implementation of the Hough Transform for real time lines detecting in the image. The proposed hardware solution includes a complete IPS with a video stream acquisition module from the camera and the output of the processed video stream in the HDMI standard. The design includes improvements in efficiency that take into account the specific properties of FPGA hardware structure. Full pipelined processing is achieved on the level of pixel by pixel image sampling. In the proposed solution, there is no need for buffering, i.e. saving the entire image frame for processing. The values of the trigonometric functions required in the algorithm were in the look-up form. Calculations are made using only integer operations. The dynamic range of the operands has been adapted to the properties of the image being processed, operations such as summation and multiplication are performed on numbers not exceeding 11 bits. The voting space is dispersed and voting for the full set of single pixel lines is performed in parallel without address collisions. The memory has been organized taking into account the size of the blocks used. The maximum frequency of image processing determined with the use of the Time Analysis tool in the Quartus II environment is 275 Mhz. This allows for real time processing of images with a resolution of 640×480 at a speed of up to 895 fps without performance drop.

***Authors**: Ph.D. Paweł Kowalski, Ph.D. Robert Smyk, Department of Control Engineering, Faculty of Electrical and Control Engineering, Gdańsk University of Technology, Gabriela Narutowicza 11/12 80-233 Gdańsk, Poland, email: pawel.kowalski@pg.edu.pl, robert.smyk@pg.edu.pl*

REFERENCES

[1] P. Mukhopadhyay and B. B. Chaudhuri, "A survey of hough transform," *Pattern Recognition*, vol. 48, no. 3, pp. 993–1010, 2015.
[2] P. V. Hough, "U.s. patent no. 3,069,654,."
[3] P. Kowalski and R. Smyk, "Wykrywanie prostych w obrazie cyfrowym z wykorzystaniem transformacji hougha," *Zeszyty Naukowe Wydziału Elektrotechniki i Automatyki Politechniki Gdańskiej*, 2018.
[4] J. Illingworth and J. Kittler, "A survey of the hough transform," *Computer vision, graphics, and image processing*, vol. 44, no. 1, pp. 87–116, 1988.
[5] X. Lu, L. Song, S. Shen, K. He, S. Yu, and N. Ling, "Parallel hough transform-based straight line detection and its fpga implementation in embedded vision," *Sensors*, vol. 13, no. 7, pp. 9223–9247, 2013.
[6] P. K. Meher, J. Valls, T.-B. Juang, K. Sridharan, and K. Maharatna, "50 years of cordic: Algorithms, architectures, and applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 9, pp. 1893–1907, 2009.
[7] M. Czyżak and R. Smyk, "Fpga computation of magnitude of complex numbers using modified cordic algorithm," *Zeszyty Naukowe Wydziału Elektrotechniki i Automatyki Politechniki Gdańskiej*, pp. 35–38, 2015.
[8] A. Elhossini and M. Moussa, "Memory efficient fpga implementation of hough transform for line and circle detection," in *2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–5, IEEE, 2012.
[9] S. Tagzout, K. Achour, and O. Djekoune, "Hough transform algorithm for fpga implementation," *Signal Processing*, vol. 81, no. 6, pp. 1295–1301, 2001.
[10] J. Guan, F. An, X. Zhang, L. Chen, and H. J. Mattausch, "Real-time straight-line detection for xga-size videos by hough transform with parallelized voting procedures," *Sensors*, vol. 17, no. 2, p. 270, 2017.
[11] S. Malmir and M. Shalchian, "Design and fpga implementation of dual-stage lane detection, based on hough transform and localized stripe features," *Microprocessors and Microsystems*, vol. 64, pp. 12–22, 2019.
[12] Z. Dong, T. Hu, R. Fuchikami, and T. Ikenaga, "Encoding-free incrementing hough transform for high frame rate and ultra-low delay straight-line detection," in *2021 17th International Conference on Machine Vision and Applications (MVA)*, pp. 1–4, IEEE, 2021.
[13] J. M. Prewitt *et al.*, "Object enhancement and extraction," *Picture processing and Psychopictorics*, vol. 10, no. 1, pp. 15–19, 1970.
[14] P. Kowalski and R. Smyk, "Sprzętowa implementacja transformacji hougha w czasie rzeczywistym," *Poznan University of Technology Academic Journals. Electrical Engineering*, pp. 45–55, 2021.