Features of the hardware implementation of real time Hough transform on FPGA

Eleonora Dorofeeva, Danila Nikiforovskii, and Ivan Deyneka

Research Institute of Light-Guided Photonics,
ITMO University, Saint-Petersburg, Russia
{eleonora.dorofeeva,danikiforovskii,igdeyneka}@corp.ifmo.ru
http://sf.ifmo.ru/

Abstract. Hough transform is a widely used computer vision algorithm. It allows to recognize objects that can be defined by the parametric function in digital images. In real-time machine vision systems that analyze video stream from digital cameras the time of processing of a single frame is extremely important. The hardware implementation of the algorithm on Field-Programmable Gate Array (FPGA) reduce the processing time by using parallel computing and quick access to embedded memory. However, the hardware implementation has a number of features. To achieve the results, the structure and principles of the FPGA operation have to well known as well as principles of FPGA interaction with the environment.

Keywords: FPGA, Hough transform, Intel FPGA, Image processing

1 Introduction

Computer vision algorithms are used in many fields of industry [1]. The most popular applications of the technology are quality control at the production stages, face detection and recognition of text or graphical information [2]. Video analysis systems are installed on fast moving platforms such as quadrocopters, cars, and trains. Thus, the image processing speed becomes the major parameter. Usage of an FPGA is an effective mechanism to highspeed image processing [3].

In the work, the FPGA-system is used as the main hardware platform for solving the processing time issues by performing the parallel computations. Hough transform is used as the main image analyzing algorithm. Methods of the algorithm optimisation and features of hardware implementation are described. The new contribution of this work compared to the previous approaches ([4], [5]) is the presented VHDL code of implementation of the Hough space, the described method for real-time circle detection using FPGA and the proposed method of the algorithm optimisation for a specific task.

2 Algorithm description

Hough transform is a technique, which used to find geometry shapes. The method involves three main steps [6]. At the first step, the shape is represented by the formula. For example, the normal form of a straight line that passing through the point with coordinates (x, y) is equal to:

$$\rho = x * \cos\theta + y * \sin\theta, \tag{1}$$

where ρ is the distance from the origin to the closest point on the straight line, and θ is the angle between the x axis and the line connecting the origin with that closest point.

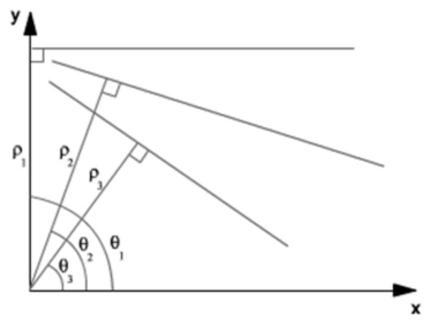

Fig. 1. Straight lines representation.

At the next step, each point related to the edges of the objects participates in the voting procedure. During the voting procedure, a voice is passed to the parameter space at the address corresponding to the current parameters. The voice may be either 1 or a value that depends on some other parameters.

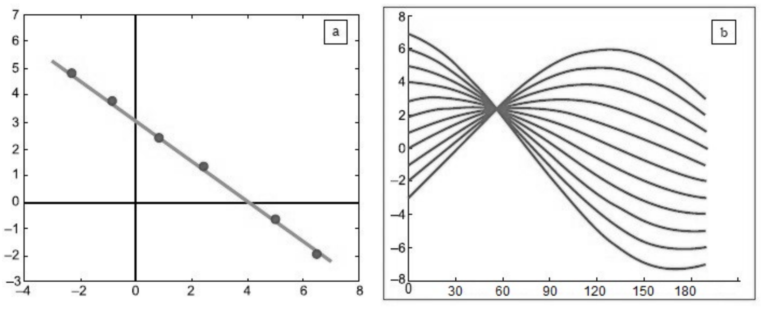

Fig. 2. Voting procedure representation. a - coordinates space; b - parameters space.

At the last step, the local maxima are defined, the coordinates of which correspond to the parameters of the most probable objects functions in the image.

3 System description

Machine vision systems contain three major elements such as digital camera, a computer and an output device or a controller of robotic parts of the system [7]. The system prototype that was organized for the work is shown in the Fig. 3.

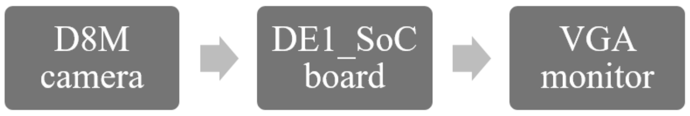

Fig. 3. The system data-flow scheme.

It contains the next elements:

- D8M camera made by Terasic;
- DE1 SoC board made by Terasic with an FPGA Cyclone V made by Intel FPGA (prev. Altera) on it;
- VGA monitor.

The stream of frames, each size of 1920 x 1080 pixels, goes from the camera to the FPGA pins. The sequence of pixels is formed by the Bayer pattern [8] (Fig. 4). The pipelining of the following calculations maximizes the capacity.

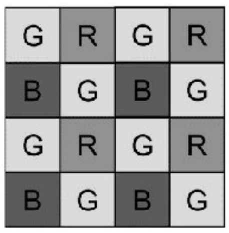

Fig. 4. The Bayer pattern.

The system operates at 65 MHz. A line change and a frame refresh take place when a horizontal or vertical signal appears. These signals are provided by VGA standard.

At the first step of the algorithms hardware implementation, the stream from the camera was converted into the three-row stream. Each row corresponds to a color from the RGB model. This step is required by the VGA standard for correct output. The interpolation method is used to complete a set of red, green, and blue values for each pixel. Registers, which can be described using VHDL (VHSIC (Very high speed integrated circuits) Hardware Description Language) should be used to get the information about neighboring pixels. The information about pixels from the previous row can be obtained by using linear buffers (Fig. 5). In the work, the issue was solved by using IP-cores that provide access to the 2-ports RAM, and by using the manually written component that was made for this purpose.

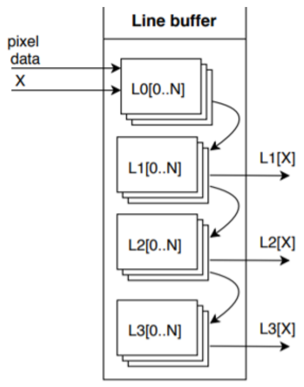

Fig. 5. The linear buffer.

The component implemented as state-machine. In each state, the current value of pixel intensity is written to one line in RAM and the values of the three previous lines with addresses X are captured. When the X value reaches the line length, the state changes and the ”oldest” written line is cleared, and becomes the line for writing.

The Canny edge detector is used to identify the pixels that belong to the objects borders [9]. It involves the next steps:

1. Gray-scale the image.
2. Apply Gaussian filter to smooth the image in order to remove the noise.
3. Find the intensity gradients of the image.
4. Apply non-maximum suppression to get rid of spurious response to edge detection.
5. Apply double threshold to determine potential edges.

In order to gray-scale the image, the information from the Y-channel from the YCbCr model can be used [10]. It provides the information about pixel intensity that can be calculated by the next equation:

$$Y = R * 0.299 + G * 0.587 + B * 0.114, \tag{2}$$

where R, G and B - values of red green and blue channels.

The coefficients in formula (2) are floating-point numbers. There are IP-cores, which perform mathematical operations with floating-point numbers, but usually, they slow down the algorithm because it takes several clock cycles to calculate. To avoid the usage of floating-point numbers in formula (2), at first, the coefficients values should be increased several times. Because of this step, the subsequent rounding does not affect accuracy. After rounding, the mathematical operations with integers can be performed. When all the operations are done, the result should be divided by the same number, which it was multiplied by at the beginning. In the work, the result is obtained by multiplying all the values by 256 and right-shifting by 8 bits, which is equivalent to dividing by 2 to the power of 8 (3).

$$Y = \frac{R * 77 + G * 150 + B * 29}{256}. \tag{3}$$

So the result can be obtained with the same precision but in just one clock cycle.

In the work, the matrix operator that shown in (4) expression is used as the Gauss filter.

$$B = \frac{1}{16} * \begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{vmatrix} * A. \tag{4}$$

To calculate the new intensity of the pixel in the center, the information from the last three rows is used. It can be obtained by using line-buffers, which was described before.

The gradient value and direction are calculated by the next equations:

$$G_x = \begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix} * A; \qquad G_y = \begin{vmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{vmatrix} * A, \tag{5}$$

where Gx and Gy (gradient values in horizontal and vertical directions) are integers, and their values are in the range from 0 to 255. In the work, all the possible variants of the squared Gx and Gy are calculated in advance and placed as initial values in the M10K memory blocks. During the system operation, mathematical operations are not used to obtain a specific value of Gx or Gy squared. The value of gradient is sent as a memory address, and the return value is a square value.

The value of the gradient direction (θ) is rounded to four main directions (0, 45, 90 and 135 degrees) and can be found using the equation (6):

$$\theta = \arctan(\frac{G_y}{G_x}). \tag{6}$$

Again, the usage of trigonometric functions slows down the processing. Therefore, in the work the M10K memory blocks and pre-calculating values are used again. The address is the concatenated (Gx|Gy) signal. The value of the memory cell in this address is a number in the range from 0 to 3 that corresponds to one of main gradient directions.

The method of memory access used in the work increases the processing speed compared to the performing of the values calculation at each iteration.

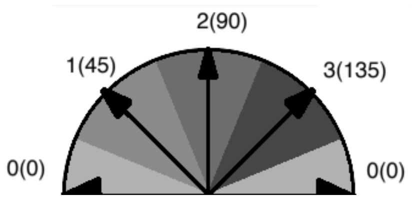

Fig. 6. The gradient directions.

4 Hough space implementation

The hardest part in the work is the implementation of the accumulator that holds the votes. The access to any cell of the parameter space should be granted in every iteration. In case of straight line detection, the space is 2-dimensional. The one dimension is an angle , which in the work is in the range from 0 to 179 in increments of 1. Another dimension is the distance in the range $[-a, d]$, where a and d are the largest side and the diagonal of the input image in pixels accordingly. In the work, the accumulator (the so-called Hough space) implemented as 180 arrays, each consisting of $(a + d)$ 10 memory blocks. Thus, the value at the address that correspond to the current pending distance is incremented in each of 180 arrays.

The M10K memory blocks were used in this paper to organize constant and immediate access to all cells of the Hough space. These blocks and instances of diastase calculating blocks are generated ones. Thus, the values of 180 different distances for 180 angles for one pixel are calculated per single cycle.
Review of VHDL code:

```
entity HoughArray is
    generic (
        X_MAX : natural := 1023;
        Y_MAX : natural := 767;
        HOUGH_SPACE_SIZE : natural := 180;
        PICTURE_SIZE : natural := Y_MAX - HOUGH_SPACE_SIZE
    );
port (  ...
        clk : in std_logic;
        reset : in std_logic;
        HS: in std_logic;
        VS: in std_logic;
        edge_info: in std_logic;
        o_hough : out std_logic_vector(9 downto 0)
    );
end entity;
```

```
architecture rtl of HoughArray is
    ...
    begin
    ...
    gen_block:
       for I in 0 to 179 generate
            rho_count_inst: rho_count
            generic map (
                cos => cosarr(I),
                sin => sinarr(I)
            )
            port map (
                ...
                x_std_vec => x_count,  -- in
                y_std_vec => y_count,  -- in
                rho => rho(I)  -- out
            );
            hough_ram_inst : hough_ram
            port map (
                ...
                data => data_sig(I),        -- in
                rdaddress => raddr_sig(I),  -- in
                wraddress => waddr_sig(I),  -- in
                wren  => wren(I),           -- in
                q => q_sig(I)               -- out
            );
    end generate gen_block;

    process (clk, reset)
    begin
    for I in 0 to 179 loop
       if (y_count < HOUGH_SPACE_SIZE) then -- build Hough space
            if (q_sig(I) < X_MAX) then      -- to avoid inferred latches
                data_sig(I) <= q_sig(I) + 1;
           else
                data_sig(I) <= (others => '1');
           end if;
           raddr_sig(I) <= rho(I);
           waddr_sig(I) <= rr_rho(I);
           wren(I) <= edge_info;
       else                            -- output Hough space
           raddr_sig(I) <= x_count;
           waddr_sig(I) <= x_count-2;
           data_sig(I) <= (others => '0');
           if (I = (y_count - PICTURE_SIZE)) then
```

```
                wren(I) <= '1';
            else
                wren(I) <= '0';
            end if;
        end if;
    end loop;

    if (rising_edge(clk)) then
        rHS <= HS;
        if (HS = 0) then
            x_count <= 0;
        else
            x_count <= x_count +1;
        end if;
        if (rHS = 0 AND HS = 1 ) then
            y_count <= y_count+1;
        end if;
        if (y_count >= PICTURE_SIZE) then
            o_hough <= q_sig(y_count-PICTURE_SIZE);
        else
            o_hough <= (others=>'0');
        end if;
    end if;
end rtl;
```

The space is build only when the y coordinate of the pixel is less than the [the picture height - 180] value, otherwise the space is displayed.Sine and cosine are calculated in advance. The certain sine and cosine for a determined angle are sent to a specific (rho) counting instance as generic numbers. The current value is used as the read address for Hough space. A read from RAM operation takes 2 clock cycles. The value that has just been read has increased by 1 and sent to the RAM at the same address.

4.1 Circle detection

In order to consider the Hough space implementation for circle detection, lets take a look to the circle radius formula, which can be calculated from its normal form:

$$r = \sqrt{(x - x_0)^2 + (y - y_0)^2}, \tag{7}$$

where (x_0, y_0) are center point coordinates, (x, y) are current point coordinates. Therefore, the Hough space should be 3-dimensional. To avoid the 3-dimensional arrays hardware implementation, circle detection can be performed of one given radius during one of 60 frames (assuming 60 fps frame rate). Thus, there will be 60 separated in time 2-dimensional spaces, each of which will have its own maxima for on specific radius, instead of one 3-dimensional space. The storage of parameters of found maxima can be easily implemented by any type of memory elements.

4.2 Advantages and disadvantages

The Hough transform algorithm has established itself as a reliable method for finding geometric shapes. The hardware implementation of the transform allows to process video in real time. The implemented algorithm is scalable due to pipeline and parallel computing. Increasing the resolution of the input image will not affect the speed of image processing. The implemented algorithm is portable due to custom-made HDL-blocks. There are no third-party IP graphics blocks used in the work. Thus, the algorithm is not related to a specific hardware. The only drawback of the Hough transform is exponential calculation time dependence on the parameters number in the formula of the detected form. It could be seen in the circle detection example. To improve the algorithm, a range and a step of parameters can be selected based on a specific task. For example, to find the road border in the picture, only lower part of the image should be processed. In addition, the diapason of line angles can be restricted, since the road will be always co-directional to the vehicle direction.

5 Conclusion

In the paper, features of hardware implementation of Hough transform were presented. Moreover, the machine vision system for real time straight line detection on Cyclone V FPGA was implemented. The method of implementation of Hough space for circles detection is considered.

Moreover, potential of the used FPGA allows to make more sophisticated projects. Completed machine vision system may be used for railway track detection, which necessary for self-driving trains.

Acknowledgments. This work was supported by the Ministry of Science and Higher Education of the Russian Federation (The unique identifier of the project: goszadanie No. 8.3134.2017/4.6)

References

1. Forsyth, D.: Computer Vision: A Modern Approach. (2004)
2. Batchelor, B.G., Whelan, P.F.: Intelligent Vision Systems for Industry. (2002)
3. Derzhanovski A.S., Sokolov S.M.: Vision data processing in real time machine vision using FPGA, `http://library.keldysh.ru/preprint.asp?id=2016-126` (2016)
4. Xin Zhou, Yasuaki Ito, and Koji Nakano: An FPGA Implementation of Hough Transform using DSP blocks and block RAMs, Japan (2013)
5. Viola P., Jones M.J.: Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2001) // Rapid Object Detection using a Boosted Cascade of Simple Features. (2001)

6. Duda, R.O., Hart, P.E.: Use of the Hough Transformation to Detect Lines and Curves in Pictures. (1971)
7. Zollhfer, M., Niegner, M., Izadi, S., Rhemann, C., Zach, C., Fisher, M.: Real-time Non-rigid Reconstruction using an RGB-D Camera // ACM Transactions on Graphics. (2014)
8. Bayer, E. B.: Color imaging array , US 3,971,065. (1975)
9. Canny, J.: A Computational Approach To Edge Detection. (1986)
10. Wright W.D.: A re-determination of the trichromatic coefficients of the spectral colours. (1928)