FPGA Implementation of a Memory-Efficient Hough Parameter Space for the Detection of Lines

David Northcote*, Louise H. Crockett, Paul Murray
Department of Electronic and Electrical Engineering, University of Strathclyde, Glasgow, Scotland, UK
*david.northcote@strath.ac.uk

***Abstract*—The Line Hough Transform (LHT) is a robust and accurate line detection algorithm, useful for applications such as lane detection in Advanced Driver Assistance Systems. For real time implementation, the LHT is demanding in terms of computation and memory, and hence Field Programmable Gate Arrays (FPGAs) are often deployed. However, many small FPGAs are incapable of implementing the LHT due to the large memory requirement of the Hough Parameter Space (HPS). This paper presents a memory efficient architecture of the LHT named the Angular Regions - Line Hough Transform (AR-LHT). We present a suitable FPGA implementation of the AR-LHT and provide a performance and resource analysis after targeting a Xilinx xc7z010-1 device. Results demonstrate that, for an image of 1024x1024 pixels, approximately 48% less memory is used than the Standard LHT. The FPGA architecture is capable of processing a single image in 9.03ms (110 frames per second).**

I. Introduction

The Line Hough Transform (LHT) is a well known technique for detecting lines within digital images [1], [2]. The LHT algorithm achieves line detection, using a simple local peak detection process in an associated Hough Parameter Space (HPS). In the HPS, the parameters of edge pixels, i.e. points in an image with sharp intensity changes, are treated as 'votes', and accumulated. The effect is to create local peaks in the HPS, corresponding to lines in the image. The algorithm is robust and can operate successfully in noisy images and when there are gaps in a line.

Major drawbacks of the LHT are its demands in terms of computation and memory. Previous work provides optimisations to reduce its complexity in software implementations; for instance, in [3] the gradient orientation is used to approximate the direction of lines. The Kernel Hough Transform [4] reduces votes by clustering pixels and limiting their votes to a small area of the HPS. Other schemes, such as the Random Hough Transform [5], Probabilistic Hough Transform [6], and Progressive Probabilistic Hough Transform [7], use random sampling to reduce the number of voting edge pixels.

Hardware implementations, using Field Programmable Gate Arrays (FPGAs), have been investigated for their ability to accelerate the LHT through parallel processing. In [8], the authors implemented a high-speed parallel FPGA architecture capable of processing a video frame of 1024x768 pixels in 5.4ms. In [9], a low-latency LHT architecture for an image of 512x512 pixels is capable of applying the LHT in 1.07ms. A pipelined, shift-and-add architecture was described in [10], capable of applying the LHT to an image of 1024x768 pixels, and accurately detecting lines in 15.59ms.

In each of these implementations, the memory consumption of the HPS was significant, making them unsuitable for small FPGAs. For example, in [9] the memory required to implement the LHT architecture is 3,317,760 bits. However, a Xilinx xc7z010-1 device is unable to host this architecture as it only contains 2,211,840 bits of dedicated FPGA memory.

The HPS can be optimised using the Adaptive Hough Transform [11]. However, this approach requires random memory access, for which an FPGA is not suited unless the candidate image is stored in on-chip memory. The authors in [12] achieved a memory efficient HPS. We recognise that this work could be further improved in terms of memory utilisation.

The original contribution of this paper is an algorithm for a memory-efficient HPS, based on a modified version of the algorithm presented in [12], and its FPGA architecture. We demonstrate that this method significantly reduces memory requirements compared to the Standard LHT [2] and other previously reported work, whilst maintaining the integrity of data stored in the HPS and overall pixel throughput.

The remainder of this paper is organised as follows: Section II summarises the Standard LHT algorithm and memory requirements of the HPS; Section III presents the proposed method; Section IV describes the FPGA architecture and simulation results; Section V evaluates the hardware architecture and provides a comparison with previously published work; and the paper is concluded in Section VI.

II. The Line Hough Transform

The LHT operates on binary images obtained after applying edge detection operators such as Sobel, Prewitt or Canny to determine the collinearity of edge pixels and thus to detect lines [13]. Each edge pixel (x, y) has an associated magnitude of displacement ρ, and orientation of displacement θ, from the image origin. These are related by the equation

$$\rho = xcos(\theta) + ysin(\theta) \quad (0^\circ \leq \theta^\circ < 180^\circ). \tag{1}$$

The HPS represents an accumulator array $A(\theta, \rho)$, that is pre-allocated in memory. The HPS is quantised with N_θ and N_ρ levels, chosen according to the applicable accuracy requirements and memory constraints. The edge pixels within the image are processed using (1) over a discrete range of θ values $[0 : \delta_\theta : \theta_{max}]$, where δ_θ is the discretisation step and θ_{max} is the maximum value of θ. The resulting parameters form an address that applies a vote to the corresponding location in the HPS.

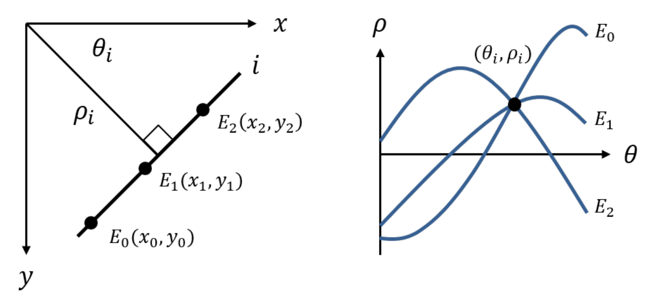


Fig. 1: LHT mapping between edge image (left) and HPS (right).

The result of voting is to create a sinusoid in the HPS for each edge pixel. Figure 1 demonstrates this relationship using three edge pixels on a line, $E_0(x_0, y_0), E_1(x_1, y_1), E_2(x_2, y_2)$, and their corresponding Hough Parameters as calculated using (1). The sinusoids produced by edge pixels that exist on line i within the image intersect at a common point (θ_i, ρ_i).

After the voting process is complete, the HPS is examined for peaks. Parameter coordinates with a significant number of votes are identified, as these correspond to linear elements in the image. Reconstruction of the lines in the spatial image domain is carried out using the inverse function of (1).

A. *HPS Memory Requirements*

The memory requirements for the HPS using the Standard LHT can be easily calculated. The HPS is restricted in size using the maximum displacement of a line ρ_{max}. This is calculated for a $W * H$ pixel image in (2).

$$\rho_{max} = \sqrt{\left(\frac{W}{2}\right)^2 + \left(\frac{H}{2}\right)^2} \tag{2}$$

This restricts the displacement parameter ρ in the accumulator array to the range $[-\rho_{max}, \rho_{max}]$.

It is necessary for the HPS to store the votes of all edge pixels, therefore the limit of the third dimension of the HPS accumulator array ('votes') is set to the length of the image diagonal, $2\rho_{max}$. The equation to calculate the total number of bits required to store the HPS in memory is therefore

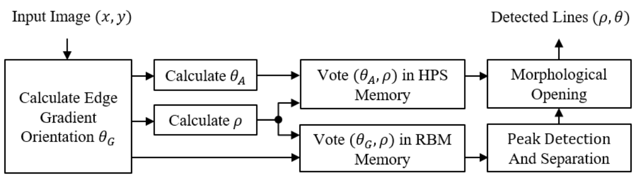
$$A_{bits} = N_\rho N_\theta \lceil log_2(2\rho_{max}) \rceil. \tag{3}$$

For example, the memory required by the HPS, for an image of 1024x1024 pixels, is 2,871,000 bits when $N_\theta = 180$ and $N_\rho = 2\lceil \rho_{max} \rceil$. We will later use (3) to compare memory utilisation of the Standard LHT with our proposed method.

III. The Proposed Method

In this section, we develop a memory-efficient design for the HPS, suitable for a small FPGA. We name this method, the Angular Regions - Line Hough Transform (AR-LHT). The AR-LHT is based on a modified version of the algorithm presented in [12], wherein the ρ axis on the HPS was reduced by partitioning the input image into subregions. Our innovation is to instead reduce the size of the HPS along the θ axis, recognising the potential to increase memory efficiency and reduce the complexity of the FPGA architecture.

The algorithm for the AR-LHT follows the procedure outlined in Figure 2. Two memories are pre-allocated, one for the HPS to accumulate votes, and another known as a Region Bitmap (RBM). This follows the general method and terminology of [12] to maintain a record of the edge gradient orientation θ_i, and displacement ρ_i, for each vote.



Input Image (x, y)
Detected Lines (ρ, θ)
Calculate Edge Gradient Orientation θ_G
Calculate θ_A
Calculate ρ
Vote (θ_A, ρ) in HPS Memory
Vote (θ_G, ρ) in RBM Memory
Morphological Opening
Peak Detection And Separation

Fig. 2: Overview of the AR-LHT algorithm.

The remainder of this section details the AR-LHT and the functionality of each of the blocks shown in Figure 2.

A. *Calculating the Edge Gradient Orientation*

The AR-LHT uses the edge gradient orientation θ_G of the candidate image to approximate the direction of lines. This reduces the number of votes that are applied to the HPS by minimising the range over which θ operates.

θ_G is calculated by initially finding the directional gradients of a greyscale image using the horizontal and vertical Sobel filters. These filters are applied to the image through convolution, deriving approximations for the horizontal and vertical gradients, G_x and G_y, for each pixel of the input image. This operation is normally carried out to derive edge pixels, produced by thresholding the gradient image G, i.e.

$$G = \sqrt{G_y^2 + G_x^2}. \tag{4}$$

The edge gradient orientation can then be computed using

$$\theta_G = \tan^{-1} \frac{G_y}{G_x}. \tag{5}$$

The value of θ_G is brought into the limits of (1) by adding 180° if $\theta_G < 0$. It can then be used in (1) as θ to calculate ρ.

B. *Optimising the HPS*

Peaks that are formed in the HPS are usually thinly dispersed throughout the array. We can use this knowledge to reduce the total number of addressable locations in the HPS.

The size of N_θ is reduced by an integer factor R. Doing so results in an accumulator array that contains $N_\rho N_\theta / R$ addressable memory locations. The number of bits required to store this reduced-size HPS is given in (6).

$$A_{bits}^{opt} = \frac{N_\rho N_\theta \lceil log_2(2R\rho_{max}) \rceil}{R} \tag{6}$$

Figure 3 provides a graph of the HPS for both the Standard LHT and the proposed method. Our investigations found that when $R > 4$ and $N_\theta = 180$, the extraction of peaks that correspond to lines in an image is no longer successful. Therefore R is limited to an integer in the range $1 < R \leq 4$.

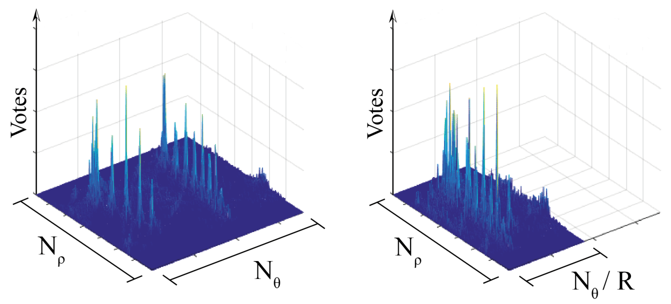


Fig. 3: The HPS for the Standard LHT (left), and the AR-LHT (right).

Votes can only be applied to the reduced HPS if they are in the range $[0, (\theta_{max} + \delta_\theta)/R - \delta_\theta]$. To ensure each vote is applied to the HPS, inapplicable votes are adjusted into the above range. This adjusted orientation is named θ_A and, while voting, is applied to the HPS with the same value of ρ.

C. Calculating the Adjusted Orientation θ_A

To obtain θ_A, we first need to carry out the integer division $\lfloor \theta_G/((\theta_{max} + \delta_\theta)/R) \rfloor$. This generates a positive integer that is used to indicate the number of times $(\theta_{max} + \delta_\theta)/R$ should be subtracted from θ_G to adjust it into the range $[0, (\theta_{max} + \delta_\theta)/R - \delta_\theta]$. The value of θ_A can then be calculated as in (7).

$$\theta_A = \theta_G - \left\lfloor \frac{\theta_G}{(\theta_{max} + \delta_\theta)/R} \right\rfloor \frac{\theta_{max} + \delta_\theta}{R} \tag{7}$$

D. Separating Peaks using the RBM

The RBM is an array of size N_θxN_ρx1, that is set to zero before processing begins. The RBM is required to keep a record of θ_G, and ρ, for all of the edge pixels that have been processed using (1). It achieves this by using (θ_G, ρ) as an index, and changing the value of that location to a binary '1'.

Upon processing an entire image, the RBM is used to cross-check the parameters of peaks found in the HPS, so that they are of the correct directional orientation before reconstruction. This is performed by searching the RBM at intervals of $(\theta_{max} + \delta_\theta)/R$ for the extracted value of θ_A.

Due to noise pixels in the edge image, noise will also exist in the RBM and could produce incorrect results. A simple method of suppressing noise in the RBM is by filtering it with a morphological opening. This is a fast method of overcoming this problem and is suitable for FPGA implementation.

IV. Implementation

In our FPGA implementation $\delta_\theta = 1^\circ$, $\theta_{max} = 179^\circ$ and $R = 4$. An overview of the FPGA architecture for the proposed method can be seen in Figure 4. The input RGB image is initially converted to greyscale for Sobel processing.

Throughout the remainder of this section, we will focus primarily on the memory aspects of our hardware design. In particular, the architecture for the HPS and RBM memories.

To correctly index the HPS memory array, the angle of θ_G must be in the range of $[0, (\theta_{max} + \delta_\theta)/R - \delta_\theta]$. As previously discussed, the practical limit of R is 4 when N_θ is 180. A simple hardware architecture designed to adjust θ_G into the above range can be seen in Figure 5.

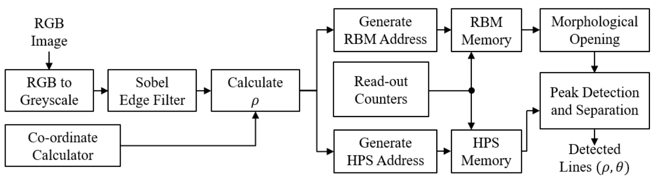


Fig. 4: Overview of the FPGA architecture.

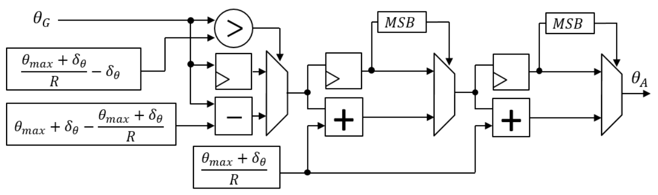


Fig. 5: Architecture to adjust θ_G into $[0, (\theta_{max} + \delta_\theta)/R - \delta_\theta]$.

The Hough parameters, ρ and θ_G, are used to generate an address to access the RBM and update it accordingly. Similarly the HPS uses ρ and the adjusted angle θ_A to generate an address to accumulate votes.

The HPS memory uses 36Kb Block RAM (BRAM) tiles, configured as dual-port 2K memory. The memory has two modes of operation, *Vote Accumulation* and *Read-out and Reset*. During *Vote Accumulation*, port A of the BRAM is configured to read the HPS vote so that it may be incremented by 1. Port B writes the updated vote back to memory.

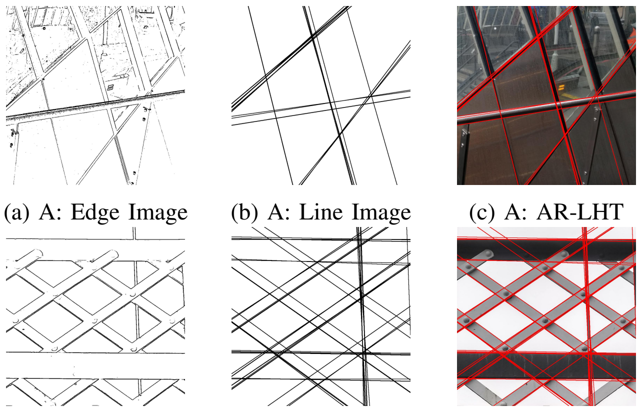
The RBM memory uses 36Kb BRAM tiles, configured as single-port 32K memory. The RBM memory has two modes. One for *Reading Parameters* and another for *Read-out and Reset*. During *Reading Parameters* the RBM sets applicable address locations defined by (θ_G, ρ) to a binary '1'.

When both memories are in *Read-out and Reset* mode, the RBM is filtered using a morphological opening while the HPS is thresholded for peaks. Both memories are then compared to cross-check line orientation. Since the HPS is R times smaller than the RBM, the HPS can be read from memory R times and compared to the filtered RBM over $(\theta_{max} + \delta_\theta)/R$ intervals. On the final iteration, the HPS is reset.

The above architecture was developed using Xilinx System Generator. Simulations correctly produced Hough Parameters corresponding to lines in several test images. These were reconstructed and superimposed on the original images for inspection; two examples are shown in Figure 6. The output images are similar to results produced using the Standard LHT.

V. Performance and Resource Analysis

The AR-LHT was synthesised and implemented using Vivado Design Suite for various image resolutions, and shown to achieve a clock frequency of 145MHz on a Xilinx xc7z010-1 device. Table I provides the memory consumption, in bits, for each tested resolution and compares them to the memory requirements of the Standard LHT, calculated using (6).


(a) A: Edge Image (b) A: Line Image (c) A: AR-LHT

(d) B: Edge Image (e) B: Line Image (f) B: AR-LHT

Fig. 6: Results of the LHT Simulation on two 1024x1024 images.

The standard LHT requires 2,871,000 bits of memory for an image of 1024x1024 pixels. The xc7z010-1 device would be unsuitable for implementation as it only contains 2,211,840 bits of dedicated FPGA memory. However, it can apply the AR-LHT to the same image resolution due to its memory-efficient HPS. The AR-LHT requires 1,474,560 bits, saving 48.64% of memory as compared to the Standard LHT.

Ser and Siu [12], developed an LHT algorithm for a reduced memory HPS. Their algorithm was applied to an image of 384x256 pixels and achieved 50% saving in memory as compared to the Standard LHT. However, our AR-LHT was able to save 58.13% when operating on the same image resolution, as shown in Table I.

Table II provides implementation results of this work and the results from other published FPGA implementations of the LHT, three of which targeted Altera devices. As Xilinx and Altera devices are not directly comparable for total resource cost, and as our work focuses on memory optimisation, we compare memory consumption and execution time.

Zhou *et al.* [9] achieved a low-latency LHT architecture for an image of 512x512 pixels, at the expense of high memory consumption. Our work, the AR-LHT, saves approximately 86% of memory for the same image size.

Chen *et al.* [14] implemented the HPS in off-chip memory, for an image of 512x512 pixels. By comparison, our AR-LHT saves approximately 87% of overall memory and adds the flexibility to cater for more than one image resolution. The execution time of this implementation would be dependant on the bandwidth and speed of the off-chip memory in use.

TABLE I: Results of this work for different resolutions, when $R = 4$, $\delta_\theta = 1^\circ$ and $\theta_{max} = 179^\circ$.

Resolution (pixels)	**Execution Time**	**Memory (Bits)**		
		AR-LHT	**Standard LHT**	**% Saved**
384x256	1.25ms	313,344	748,440	58.13
512x512	2.71ms	442,368	1,306,800	66.15
1024x768	7.01ms	1,345,536	2,534,400	46.90
1024x1024	9.03ms	1,474,560	2,871,000	48.64

Guan *et al.* [8] created a system which applied the LHT to a video stream. They reduced the total memory requirements of the HPS, by setting the discretisation step of θ to 2° ($\delta_\theta = 2^\circ$). This has the effect of reducing the accuracy of line detection, unlike our work, which still achieves $\delta_\theta = 1^\circ$. Even for unequal δ_θ, the AR-LHT provides a memory saving of 16%.

Lu *et al.* [10] developed a shift-and-add LHT architecture with δ_θ set to 0.8952°. This achieved better accuracy of extracted lines whilst increasing the memory requirements of the HPS. If this approach was combined with the AR-LHT (a possible avenue of future work), the HPS could become more memory-efficient while also improving accuracy.

VI. Conclusion

This paper has presented the AR-LHT, a memory-efficient algorithm for the detection of lines in images. It was demonstrated that the memory consumption of the HPS could be reduced by exploiting the thin dispersion of peaks within it. We found that two separate, smaller memories (a reduced-size HPS across the θ dimension, and a 1-bit RBM) could be used in conjunction to substantially reduce memory utilisation compared to the Standard LHT. After voting was complete, the RBM was used to determine the true orientation of detected peaks in the reduced HPS. The incorporation of a morphological opening post-processing stage (applied to the RBM) was crucial in the success of the proposed method.

The AR-LHT algorithm was developed using Xilinx System Generator. A series of simulations were carried out on a set of test images. Results demonstrated that the output Hough Parameters correlated successfully with lines in the original test image. The hardware architecture was then synthesised and implemented using Xilinx tools, resulting in very efficient resource requirements in comparison to similar work. An image of 1024x1024 pixels was found to save 48.64% of memory compared to the Standard LHT. The FPGA architecture was capable of operating at 145MHz, and can process the image in 9.03ms, equivalent to a frame rate of 110fps.

TABLE II: Results of the proposed method and comparison with related works

Algorithm	**Device**	**Resolution**	**Frequency**	**Execution Time**	**Logic Blocks**	**DSPs**	**Memory (Bits)**
Proposed Method (AR-LHT)	Xilinx XC7Z010-1	1024x1024	145MHz	9.03ms (110fps)	2202 (Slices)	0	1,474,560
Zhou *et al.* [9]	Xilinx XC6VLX240T-1	512x512	200MHz	1.07ms (936fps)	14493 (Slices)	178	3,317,760
Chen *et al.* [14]	Altera EP2S180F1508C3	512x512	200MHz	2.07-3.61ms (277-483fps)	855 (ALUTs)+ 421 (Registers)	0	3,270,032 (off-chip)+ 223,360 (on-chip)
Guan *et al.* [8]	Altera EP4SGX230KF40C2	1024x768	200MHz	5.4ms (185fps)	1737 (ALUTs)+ 1196 (Registers)	32	1,565,202 ($\delta_\theta = 2^\circ$)
Lu *et al.* [10]	Altera EP4CE115F29	1024x768	200MHz	15.59ms (64fps)	15,704 (LEs)	8	3,052,544

References

[1] P. V. C. Hough, "Machine analysis of bubble chamber pictures," in *Proceedings, 2nd International Conference on High-Energy Accelerators and Instrumentation, HEACC*, CERN, Geneva, Switzerland, Sep. 1959, pp. 554–558.

[2] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Commun. ACM*, vol. 15, no. 1, pp. 11–15, Jan. 1972.

[3] F. O'Gorman and M. B. Clowes, "Finding picture edges through collinearity of feature points," *IEEE Transactions on Computers*, vol. C-25, no. 4, pp. 449–456, April 1976.

[4] L. A. F. Fernandes and M. M. Oliveira, "Real-time line detection through an improved Hough transform voting scheme," *Pattern Recognition*, vol. 41, no. 1, pp. 299–314, 2008.

[5] L. Xu, E. Oja, and P. Kultanen, "A new curve detection method: Randomized Hough transform (RHT)," *Pattern Recognition Letters*, vol. 11, no. 5, pp. 331–338, 1990.

[6] N. Kiryati and Y. Eldar, "A Probabilistic Hough Transform," *Pattern Recognition*, vol. 24, no. 4, pp. 303–316, 1991.

[7] J. Matas, C. Galambos, and J. Kittler, "Robust Detection of Lines Using the Progressive Probabilistic Hough Transform," *Computer Vision and Image Understanding*, vol. 78, no. 1, pp. 119–137, 2000.

[8] 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 (Switzerland)*, vol. 17, no. 2, 2017.

[9] X. Zhou, Y. Ito, and K. Nakano, "An FPGA Implementation of Hough Transform using DSP blocks and block RAMs," *Bulletin of Networking, Computing, Systems, and Software*, vol. 2, no. 1, pp. 18–24, January 2013.

[10] 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 (Basel, Switzerland)*, vol. 13, no. 7, pp. 9223–9247, 2013.

[11] J. Illingworth and J. Kittler, "The Adaptive Hough Transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 5, pp. 690–698, 1987.

[12] P.-k. Ser and W.-c. Siu, "Memory compression for straight line recognition using the Hough transform," vol. 16, no. February, pp. 133–145, 1995.

[13] R. Gonzalez and R. Woods, *Digital Image Processing*. Pearson Education, 2011.

[14] Z. H. Chen, A. W. Y. Su, and M. T. Sun, "Resource-efficient fpga architecture and implementation of hough transform," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 8, pp. 1419–1428, Aug 2012.