Parallel Multi-Sector Algorithm for the General Hough Transform

Emeric K. Jolly and Martin Fleury
Electronic Systems Engineering Dept.
University of Essex,
Colchester, CO4 3SQ, United Kingdom
e-mail: {ekjol;fleum}@essex.ac.uk

Abstract

The Multi-Sector Algorithm (MSA) is a simplification of the CORDIC algorithm to more closely meet the requirements for a real-time general Hough transform applications. The MSA can form a pipeline, and multiple angle rotations are performed in parallel. The whole has been tested on Virtex platform FPGAs to find straight edges within images. Angular resolution is incrementally scalable. Video-rate processing is easily achieved. The ability to run independent MS units in parallel is only limited by on-chip memory restrictions.

1. Introduction

The Hough transform (HT) [11] is frequently used to locate possibly occluded straight edges or lines in machine vision. Each detected edge pixel in a binary image votes for a potential edge upon which it might lie [4]. To avoid needing infinite slopes for vertical edges, in the slope-intercept method [7], the polar coordinates of an edge pixel act as indices to the vote array. The HT is potentially suitable for video-rate applications such as motion detection (by comparison of successive HT transformed frames) [10], but the computational burden is high, motivating parallel and/or hardware implementations. Notice that if gray-level values are accumulated in the vote array then a Radon transform is accomplished [15].

Hardware designs for the general HT have applied the Coordinate Rotational Digital Computer (CORDIC) algorithm [6] [1] to the slope-intercept method. This is because the CORDIC computes, almost as a by-product, rectangular to polar coordinate transformations. The HT CORDIC method proceeds by incrementing the angle of the normal to a potential edge line segment in order to find all potential radii. For each angle, a series of convergent micro-rotations is generated. The series are usually unrolled to form a micro-rotation pipeline. Look-up table (LUTs) techniques (to establish the direction of a particular micro-rotation) and scale factor compensations [22][5] have been investigated, though these impact throughput and/or hardware complexity.

The HT is an attractive demonstrator of CORDIC techniques, such as the mixed-radix CORDIC [3]. However, it is a noticeable feature of some hardware papers on CORDIC HTs that no illustration of application to real images is given. In this paper, we question whether the repeated convergent cycles to generate each angle in the HT algorithm are the most suitable and efficient way to generate votes. Our approach arises from the observation that the angles to be generated are not random. They are an ordered series of calculations as the angle of the normal is successively incremented. Therefore, rather than a convergent series of calculations, each angle calculation can be generated iteratively. At the same time, the image origin is shifted to the center. This allows the angle space to be divided into sectors, so that calculation to proceed in parallel on a per-sector basis. As the method of angle calculation does not impact on the HT itself, the transform remains a general HT with no differences.

By simplifying the design, chip area usage is reduced. As there is no interdependence in calculations on edge pixel inputs, the design can then be replicated with further parallelism. It turns out that on current Field-Programmable Gate Array (FPGA) technology, the main limit to single chip parallelism is not the HT calculation unit (which occupies less than 7% of a Virtex-II XC2V3000 [20] working to 15-bit fixed point for 0.9^o angular resolution on an image size 256×256 pixels) but on-chip memory size to support independent vote arrays.

In our adaptation of the CORDIC idea, the micro-rotations are constant and uni-directional, while still achieving line segment detection. Re-scaling is avoided by judicious choice of fixed micro-rotation size resulting in gain approximating to one, while retaining sufficient detection accuracy for the application. The calculations within each sector can also be pipelined. With the search space di-

vided into equal sectors, we call these techniques the Multi-Sector Algorithm (MSA) and a pipelined implementation the PMSA. We demonstrate the PMSA on a standard 256×256 pixel imagery using the Xilinx Virtex-II FPGA. The worst-case throughput is about 30 frame/s for a dual PMSA, with FPGA running at 50 MHz and angular resolution 0.9^o. The angular resolution of the micro-rotations is incrementally tunable. We have implemented a version with 0.9^o resolution. The main trade-off for the enhanced speed without scaling or convergence is an approximation to the initial offset by shifts and adds and the approximation of the gain factor to one, with some (limited) loss in accuracy. The accuracy is also scalable as by running multiple PMSA in parallel, the angle range is reduced for each PMSA, resulting in reduced propagation of gain approximation error. In our design, as in others, an edge-detected binary image is input and a further peak detection stage is required, which has the effect of increasing pipeline latency. However, both edge detection and peak detection are implementable in hardware.

The remainder of this paper is organized as follows. Section 2 explains how we have simplified the CORDIC algorithm and details the MSA and PMSA method of calculating the general HT. Section 3 reports results in terms of image processing and performance. In Section 4, other implementations are briefly considered from a variety of aspects. Finally, Section 5 draws some conclusions.

2. Hardware Hough Transform

The generation of trigonometric functions (sines and cosines) is one of the main barriers to implementation of the HT in hardware. One technique is to employ LUTs [9][12], but on the FPGA implementation intended by us, LUTs would have a severe impact on on-chip memory usage, given that memory is also required for vote arrays. Another possibility is distributed arithmetic [2]. The CORDIC-based algorithm remains attractive.

2.1. CORDIC algorithm

Consider coordinate origins at the geometrical center of an image, then seek the length of a radius R from the origin normal to a straight edge passing between a detected pixel with Cartesian coordinates (X, Y). R subtends an angle θ so that

$$R = X\cos\theta + Y\sin\theta, \tag{1}$$

with θ valid from 0 to 2Π.

In practice, θ is ranged from 0 to Π as the sign of R distinguishes the position of R. Dividing through by $\cos\theta$ and with suitable trigonometric substitutions yields:

$$R_X = (X + Y\tan\theta)\cos\theta, 0 < \theta < \frac{\Pi}{2}, \Pi < \theta < \frac{3\Pi}{2} \tag{2}$$

$$R_Y = (Y - X\tan\theta)\cos\theta, \frac{\Pi}{2} < \theta + \frac{\Pi}{2} < \Pi, \frac{3\Pi}{2} < \theta + \frac{\Pi}{2} < 2\Pi, \tag{3}$$

which are also the basis of the CORDIC algorithm for hardware calculation of trigonometric values by means of micro-rotations. By varying θ, all possible values of R are found for varying straight lines through invariant (X, Y). θ can be replaced by $\arctan\phi$, so that $\tan(\arctan\phi) = \phi$.

In the traditional CORDIC [1] applied in [22][5], $\phi = \pm 2^{-k}$ with $k = 0, 1, 2....$, allowing a set of micro-rotations to converge to a given angle. However, this requires a look-up table to establish the direction of a micro-rotation at any stage in the iteration and a scaling re-adjustment [5].

2.2. Constant rotation CORDIC-like algorithm

For this application, it turns out that the micro-rotations can be uni-directional and of constant size causing a series of angles to be generated. Setting constant $k = 5$ means $\phi = 2^{-5}$ and the micro-rotation step-size is 1.79^o. In Section 3.2 $k = 6$, $\phi = 2^{-6}$ and angular resolution is 0.9^o.) The terms multiplied by 'tanθ' in (2) and (3) will result in a simple right shift operation by six places if $k = 6$. The sequence of constant size micro-rotations is defined as:

$$R_{1X}(m+1) = R_{1X}(m) + R_{1Y}(m)\phi \tag{4}$$

$$R_{1Y}(m+1) = R_{1Y}(m) - R_{1X}(m)\phi, \tag{5}$$

with $m = 0, 1, \ldots, M_{step}$, M_{step} being the number of micro-rotations. The gain, $\cos(\arctan\phi)$, approaches one for sufficiently small ϕ. However, if all is unchanged, smaller angular resolution will result in more micro-rotations to cover the angular range and increased propagation of error as m approaches M_{step}. Therefore, there is a trade-off in accuracy between gain and number of micro-rotations. Accuracy can be gained without any trade off by decreasing the range of each set of micro-rotations (see Section 2.5). Fig. 1 shows an example hardware cell to calculate successive values of the micro-rotation sequence.

2.3. Multi-Sector Algorithm

The MSA uses the same micro-rotations but several cells are employed to process the angle range and generate radii. A simple MSA for instance, will have one cell of the type shown in Fig. 1 for incrementing angles and a second variant cell to calculate

$$R_{2X}(m+1) = R_{2X}(m) - R_{2Y}(m)\phi \tag{6}$$

$$R_{2Y}(m+1) = R_{2Y}(m) + R_{2X}(m)\phi, \tag{7}$$

for decrementing angles, resulting in double the throughput.

The initial radii inputs to the two cells (refer to Fig. 2) are calculated from the original X and Y pixel coordinates

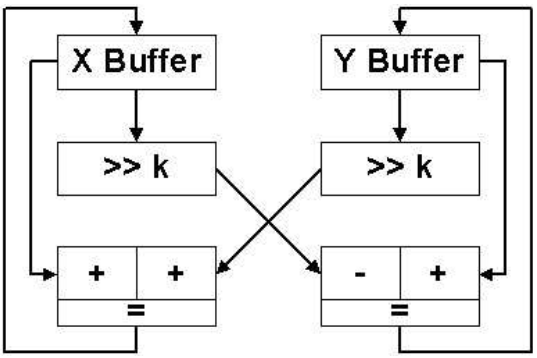

Figure 1. Example CORDIC-like cell for calculating radii values in successive micro-rotation increments.

(relative to the image center). For the simple MST, initialize these values with the radii corresponding to an offset of $\frac{\Pi}{4}$. This can be achieved with a series of shifts and additions to the original pixel coordinates. To maintain sufficient accuracy for machine vision applications the values stored in the R_{1X}, R_{1Y}, R_{2X} and R_{2Y} registers are $\frac{1}{\tan\Delta\theta} = \frac{1}{\phi}$ times larger than the actual values, for example $\frac{1}{\phi} = 2^6 = 64$. Thus, one must initialize $R_{1x} = R_{2x}$ from:

$$64 \times X\cos(\frac{\Pi}{4}) = X \times 64 \times \frac{\sqrt{2}}{2} \approx 45.25 \times X \qquad (8)$$

and equivalently for $R_{1y} = R_{2y}$. As 32 + 8 + 4 + 1 + 0.25 = 45.25, the shifts and adds applied to form (8) are $X << 5 + X << 3 + X << 2 + X + X >> 2$. The resulting bit width to store the fixed-point value of the radii is 1 (sign bit) + 8 (radii address range) + 6 (shift-add accuracy) = 15 bits.

2.4. MSA pipeline

A block diagram of an MSA is given in Fig. 3. It is possible to form a three stage pipeline consisting of: (1) Calculate radii; (2) Read and increment vote; (3) Write vote back. Therefore, a further improvement in throughput arises from a pipelined MSA (PMSA) if on-chip, dual-port block RAMs, allowing simultaneous read and writes, are employed to store votes. On-chip block RAMs have been shown [19] to be an effective addition to FPGA architecture.

2.5. Multiple PMSA

By dividing up the angular range (0–360^o) multiple PMSA result in further speedup (double for double PMSA). The calculation performed by each PMSA unit is independent but each must be initiated at different angles in the

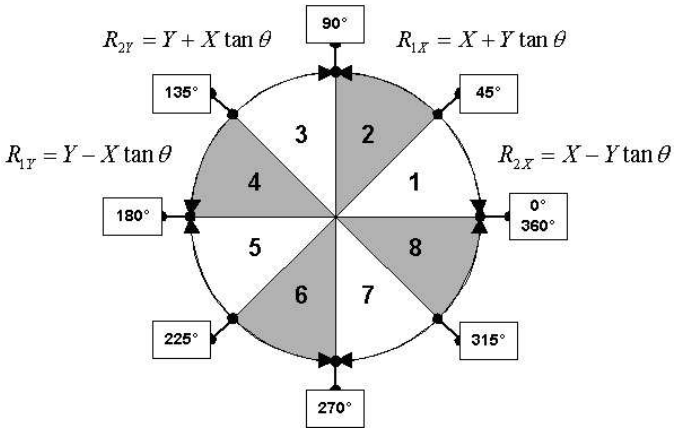

Figure 2. Division into four of the radii search range, with eight regions according to the sign of the calculated radius.

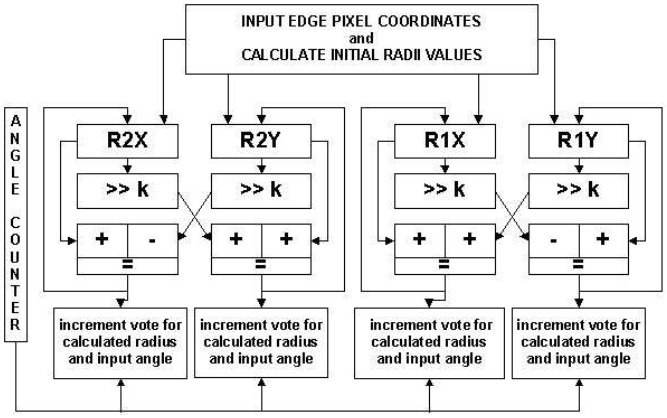

Figure 3. MSA cell for simultaneous calculation of four radii values.

range. For a double PMSA, the initial offset to the two PMSA are respectively $\frac{\Pi}{8}$ and $\frac{3\Pi}{8}$.

There is a further bonus from multiple PMSAs, namely the angle range is reduced and hence the accuracy increases as the gain error approximation does not propagate over as many micro-rotations. An error analysis of the gain proceeds by (say) taking the z-transform of equation pair (4) and (5) and writing in terms of $R_{1X}(0)$ and $R_{1Y}(0)$, which are known values.

3. Results

3.1. Image processing

The algorithm was performed on a Virtex-II XC2V3000 FPGA [20]. To check correct working, an image with a single horizontal line, Fig. 4 was employed, with the resulting voting array in Fig. 3.1. A logarithmic intensity function has been employed so that smaller votes are still visible. Notice the well-defined peaks and the characteristic sinusoidal wave pattern, arising from re-arranging (1) to form:

$$R = (X^2 + Y^2)^{\frac{1}{2}} \sin(\theta + \tan^{-1}(X/Y)) \qquad (9)$$

A standard image 256×256-pixel image, "airplane"[1],

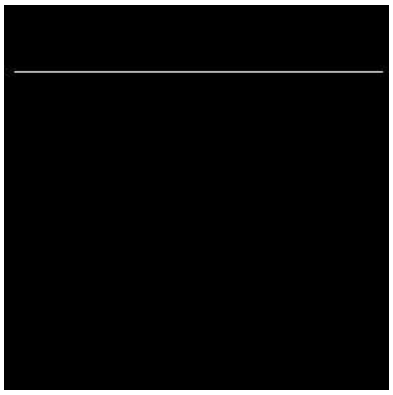

Figure 4. Horizontal line image

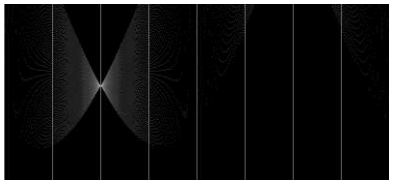

Figure 5. Vote array for Fig. 4

Fig. 6, with clear linear features, the runway borders, was then employed. A binary-thresholded edge image, Fig. 7, was the input to the MSA. The voting array is captured in Fig. 8, with the x axis running from 0 to 360^o marked off at 45^o intervals and the vertical y axis ranging from 0 to 182 pixel radius. The vertical dimension allows for the maximum radius in a 256×256 image. Fig. 9 is the line image created from the voting array. All votes above 32 have been re-created as straight lines. Therefore, it should be stressed that peak detection has *not* been applied to remove shorter edge segments. The plane itself has caused some disturbance to the picture, though the runway borders are apparent. Had the plane itself been first been removed as the result (say) of a template matching algorithm, then Fig. 10 would be the resulting vote array and Fig. 11 is the line image created from the voting array. The main weakness of the image is that the runway leading off to the right is weakly represented.

[1] Available from the USC-SIPI Image Database at `http://sipi.usc.edu/database/`.

3.2. Performance

The maximum radius is defined as:

$$R_{max} = \sqrt{\frac{X^2 + Y^2}{4}}, \qquad (10)$$

assuming the origin of coordinates is in the center. For a 256×256 pixel image, $R_{max} = 181$. The angle range is 0 to 360^o. The memory allocated for the angle range is divided into eight sectors for a single MSA (16 for a double MSA, and so on). If these sectors are mapped to different block RAMs (BRAMs) then there is independent read and write access to the voting array for each sector.

For a single MSA, the number of microrotations, M_{step}, needed to create a set of votes for a single pixel is given by

$$M_{step} = \frac{360}{N_{cell}.8.A_{res}}, \qquad (11)$$

where N_{cell} is the number of MSA cells and A_{res} is the angular resolution.

Consider the maximum number of clock cycles, C_{max}, to process an image, then

$$C_{max} = C_{init} + C_{trans}, \qquad (12)$$

where C_{init} is the time to initialize the vote array and C_{trans} is the time to perform an MSA. Now

$$C_{init} \approx (R_{max} + 1) \times M_{step} \qquad (13)$$

which for a 256×256 pixel image with angular resolution $\arctan(2^{-6})$ is $182 \times 50 = 9100$ clock cycles. If all pixels in an image are edge pixels then

$$C_{trans} \approx (C_{rescale} + M_{steps}) \times I_{size}, \qquad (14)$$

where C_{rescle} is the number of clock cycles to retrieve an edge pixel from memory, rescale the pixel to the entry angle of the MSA cells, and initialize the pipeline of the MSA. $I_{size} = X \times Y$ is the image size in pixels. $C_{rescale}$ can also run in parallel with the voting loop, thus saving some clock cycles. For the implementation running at 50 MHz on a 256×256 pixel image with angular resolution $\arctan(2^{-6}) = 0.9^o$, $C_{rescale}$ was six clock cycles.

The minimum frame rate, F_{min} is given by

$$F_{min} = \frac{S_{clock}}{C_{max}} \quad (15)$$

For the one PMSA cell implementation on a 256×256 pixel image with clock speed at 50 MHz, then $F_{min} \approx 13.5$ frame/s and with two PMSA cells $F_{min} \approx 24.5$ frame/s. If $C_{rescale}$ is run in parallel with the PMSAs then $F_{min} \approx 30.4$ frame/s. As the PMSA is compute intensive, unlike some image-processing tasks on an FPGA Virtex-II, e.g. [8], the clock width imposed by access to external interfaces is not a limiting factor.

3.3. Resource Usage

Table 1 shows resource usage on the Virtex-II FPGA in the implementation, from which it is clear that the main restriction to further parallelism is the number of dedicated block RAMs (BRAMs). The BRAMs principally store the vote arrays, and therefore optimization of vote array storage is required. Currently, 10-bit counters are used for a resolution of 0.9^o. In addition, to simplify the design, a concatenation of the radii and angular address resulted in some memory redundancy. Worst case vote counts may not be met and there are also other opportunities to reduce storage through counter quantization and a re-structured addressing scheme.

Resource	1 MSA cell	2 MSA cells
BRAM	76 out of 96	84 out of 96
Slice	1,113 (7%)	1,185 (12%)
LUT	1,987 (6%)	3,487 (12%)
Flip Flop	750 (2%)	1,160 (4%)

Table 1. Resource usage by number and percentage on Xilinx Virtex-II XC2V3000

4. Related work

In general, comparisons between the performance of HT implementations are not helpful as they cannot account for changes in technology, the angular resolution, or variations in the algorithm. For example, if the modified HT [14] is performed then a gradient image is input causing only one vote per oriented edge point. Since [14], there have been many attempts to reduce computation, such as the randomized [21] and fast HT [13]. However, the latter two algorithms are unsuitable for hardware implementation because of their irregular structure. The general HT [7] is entirely regular. It has also been shown in [22] that the modified HT [14] is capable of hardware implementation by adding units at the head of the CORDIC pipeline to calculate the gradient from two orthogonal Sobel inputs [4].

The sequential algorithm for image size $\sqrt{N} \times \sqrt{N}$ with an $n \times n$ parameter space takes $O(nN)$. There have been numerous fine-grained (systolic, SIMD mesh, and pyramid [17]) parallel versions of the HT, as well as distributed memory multiprocessors, with an extensive list in [3]. The main detraction of SIMD solutions is that hardware complexity can grow with the size of the image. For example, in [15], an $O(1)$ (constant time) algorithm on a reconfigurable mesh is reported but at a cost of n^2N processing elements. As to actual performance, in [16], a 256×256 pixel HT is reported as taking 0.13 s or 7 frame/s on the MPP SIMD machine.

A VLSI implementation of the HT, LSI Logic's L64250 chip [12] (using trigonometric LUTs) is reported to take 5.12 s for a 256×256 pixel image, which does not appear to be competitive with an FPGA implementation except perhaps in power usage.

Turning to CORDIC-based implementations, in [22], a 256×256 pixel image is estimated to take between 37.5 and 87.5 ms, or 26 and 11 frame/s but the design was not apparently implemented. In [5], the design of [22] was modified and was implemented on a Xilinx XC4010XL-PC84 FPGA. The 16-bit fixed-point version over 12 iterations was able to transform a 128×128 pixel image with a resolution of 1.4^o at 38 frame/s. The clock speed realized was 40 MHz.

A number of variants to the CORDIC algorithm have been proposed. In [3], a radix-4 CORDIC is considered. As a radix-4 CORDIC requires a non-constant scaling factor — unlike the traditional radix-2 version, a scheme to mix radix-2 with radix-4 was devised to avoid the scaling problems. [3] also consider parallelization by splitting the angle space. In [18], two processors are used, one to calculate the angular increment and the other to complete the calculation of the radius. As in our solution, the origin of coordinates is at the image center. In [18], the angular space is divided into eight sectors, allowing parallel computation of points in each sector, by manipulation of the phase factor $\tan^{-1}(X/Y)$ in (9). However, the number of sectors exploitable by the technique in [18] is restricted to eight.

5. Conclusion

Generation of the voting array is one of the principal obstacles to achieving a real-time general Hough Transform (HT) in hardware. This is because of the need to calculate trigonometric values. The CORDIC algorithm is one method that has already been applied to the HT on a reconfigurable FPGA. FPGAs are well suited to image-processing applications offering hardware parallelism and fine-grained processing. However, in this paper we con-

sider how the basic CORDIC idea can be modified to improve the throughput through constant-size micro-rotations. The design in this paper used pipelining of edge coordinate inputs and vote generation. Parallelism is also present by simultaneous calculation of pixels within image sectors. As the complexity of the design is much reduced over a pure CORDIC scheme the actual restriction on platform FPGAs is the number of on-chip RAM blocks. As a significant problem is the size of the vote array, further work is to consider reducing this requirement by (say) quantization. Finally, a full error analysis is of interest for applications for which this is appropriate.

References

[1] R. Andraka. A survey of CORDIC algorithms for FPGA based computers. In 6^{th} *ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'98)*, pages 191–200, 1998.

[2] F. Bensaali, A. Amira, and A. Bouridane. An efficient architecture for color space conversion using distributed arithmetic. In $\underline{14^{th}}$ *Conference on Field Programmable Logic and Applications (FPL 2004)*, pages 991–995, 2004. LNCS No. 3203.

[3] J. Bruguera, N. Guil, T. Lang, J. Villalba, and E. Zapata. Cordic based parallel/pipelined architecture for the Hough transform. *Journal of VLSI Signal Processing*, 12:207–221, 1996.

[4] E. Davies. *Machine Vision: Theory, Algorithms, Practicalities*. Academic Press, San Diego, CA, 2nd edition, 1997.

[5] D. Deng and H. ElGindy. High-speed parameterisable Hough transform using reconfigurable hardware. In *Pan-Sydney Area Workshop on Visual Information Processing (VIP2001)*, pages 51–57, 2001.

[6] E. Deptrettere, P. Dewilde, and R. Udo. Pipelined CORDIC architecture for fast VLSI filterig and array processing. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 41.A.6.1–41.A.6.4, 1984.

[7] R. Duda and P. Hart. Use of the Hough transform to detect lines and curves in pictures. *Communications of the ACM*, 15:11–15, 1972.

[8] M. Fleury, R. Self, and A. Downton. Multi-spectral satellite image processing on a platform fpga engine. In *Military and Aeronautics Logic Devices (MAPLD'05)*, 2005. Paper no. 133.

[9] K. Hanahara, T. Maruyama, and T. Uchiyama. A real time processor for the Hough transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(1):121–125, 1988.

[10] M. Hills, T. Pridmore, and S. Mills. Object tracking through Hough space. In *International Conference on Visual Information Engineering (VIE'03)*, 2003.

[11] P. C. Hough. Method and means for recognizing complex patterns, 1962. US Patent 3069654.

[12] Innovasic Semiconductor, Albuquerque, New Mexico. IA64250 histogram/Hough transform processor data sheet, 2000. The IA64250 is compatible with LSI's L64250.

[13] H. Li, M. Lavin, and R. Le Master. Fast Hough transform: A hierarchical approach. *Computer Vision, Graphical and Image Processing*, 36:139–161, 1986.

[14] F. O'Gorman and M. Clowes. Finding picture edges through collinearity of feature points. *IEEE Transactions on Computers*, 25:449–456, 1976.

[15] Y. Pan, K. Li, and M. Hamdi. An improved constant-time algorithm for the Radon and Hough transforms. *IEEE Transactions on Systems, Man, and Cybernetics*, 29(4):417–421, 1999.

[16] A. Rosenfeld, J. Omelas, and Y. Hung. Hough transform for mesh-connected simd parallel processors. *Computer Vision, Graphical and Image Processing*, 41(3):293–305, 1988.

[17] S. Shoari, A. Kavianpour, and N. Bagherzadeh. Pyramid simulation of image processing applications. *Image and Vision Computing*, 12(8):523–9, 1994.

[18] D. Timmerman, H. Hahn, and B. J. Hosticka. Hough transform using CORDIC transform. *Electronic Letters*, 25(3):205–206, 1989.

[19] S. Wilton, J. Rose, and Z. Vranesic. The memory/logic interface in FPGAs with large embedded memory arrays. *IEEE Transactions on VLSI*, 7(1):80–91, 1999.

[20] Xilinx Inc., San Jose, CA. *Virtex-II Platform FPGA Handbook*, 2002.

[21] L. Xu, E. Oja, and P. Kultanen. A new curve detection method: Randomized Hough transform (RHT). *Pattern Recognition Letters*, 11:331–338, 1990.

[22] F. Zhou and P. Kornerup. A high speed Hough transform using CORDIC. In *International Conference on Digital Signal Processing (DSP'95)*, 1995.

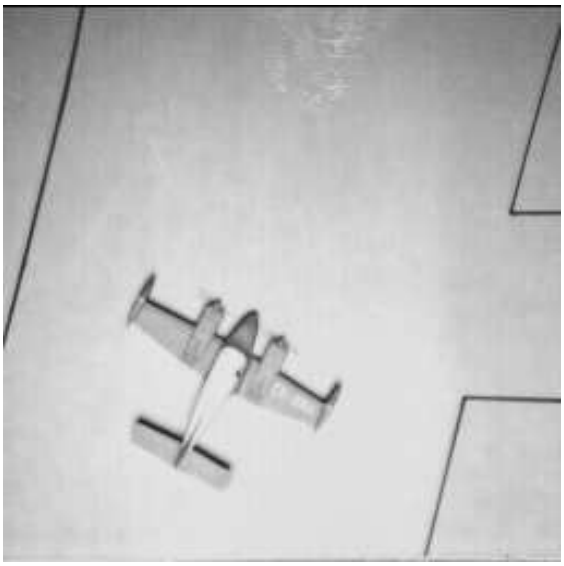

Figure 6. 256×256 **"plane" image**

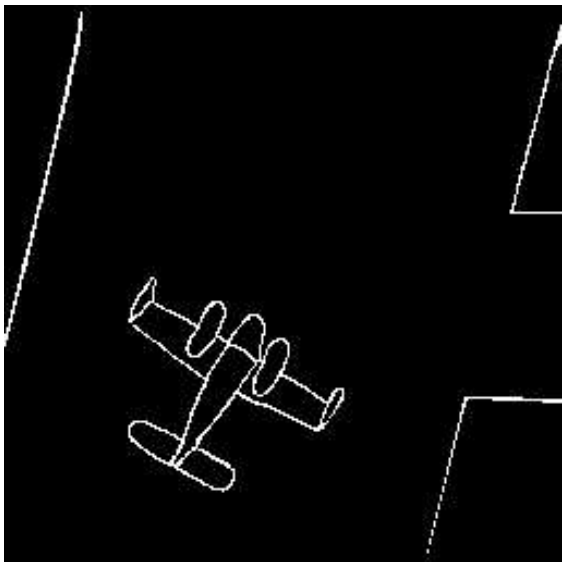

Figure 7. Thresholded edge image

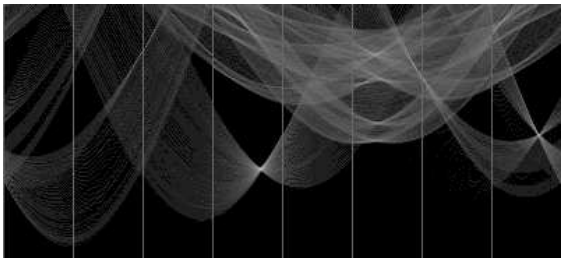

Figure 8. Voting array from Fig. 7

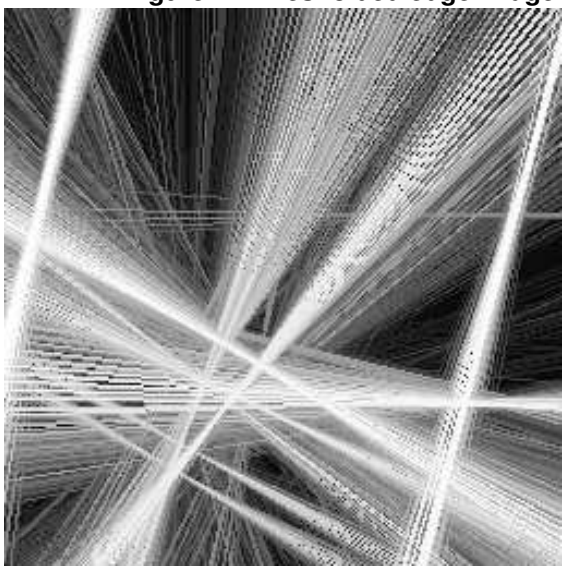

Figure 9. Line image of Fig. 7

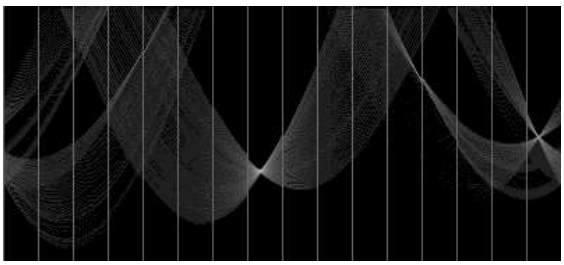

Figure 10. Voting array after prior removal of the 'plane

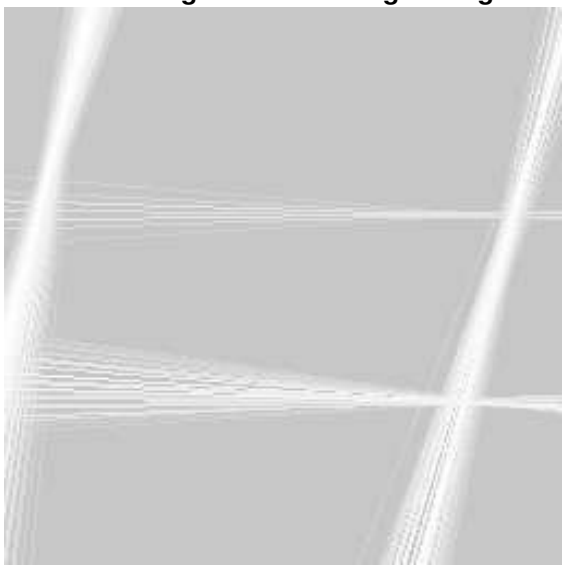

Figure 11. Line image after prior removal of the 'plane