A Compact System for Real-Time Detection of Line Segments

Nozomu Nagata and Tsutomu Maruyama

Systems and Information Engineering, University of Tsukuba,
1-1-1 Ten-ou-dai Tsukuba Ibaraki 305-8573 Japan
nagata@darwin.esys.tsukuba.ac.jp

Abstract. In this paper, we describe a compact circuit for real-time detection of line segments using the Line Hough Transform (LHT). The LHT is a technique to find out lines in an image. The LHT is robust to noises, but requires long computation time. The circuit calculates (1) r and θ of lines (r is the distance from the origin to a line and θ is the angle of the line) by the LHT units in parallel, and (2) start and end points of the lines by the other units which are completely pipelined with the LHT units. With this parallel and pipeline processing, the circuit can detects line segments by $\pi/512$ angle steps in a standard size image (640×480) in real-time. This circuit was implemented on an off-the-shelf PCI board with one Field Programmable Gate Array (FPGA) chip. The size of the circuit is 45% of the chip, which makes it possible to implement other circuits for higher level processing of object recognition on the same chip, or the performance can be improved twice by using the rest of hardware resources.

1 Introduction

Detection of line segments is a very important step in object recognition. Line segments with other informations such as distances to the planes surrounded by them will help to construct 3-D models of real-world. The Line Hough Transform (LHT) is a technique to find out r (distance from the origin to a line) and θ (the angle of the line) of lines in an image. The LHT is very robust to noises, but requires long computation time for calculating (r, θ) of all candidates. The calculation, however, is very regular and simple, and can be accelerated by parallel processing by hardware. In our circuit, r and θ of lines are detected by LHT units first, and then, start and end points of the lines are obtained by the other units which are completely pipelined with the LHT units. In these units, start and end points of lines are obtained by considering distances between edge points on the lines. This parallel and pipeline processing make it possible to find out start and end points of lines in standard size images (640×480) in real-time.

2 Previous Works

Many approaches by software to reduce the computational complexity have been proposed, and efficient algorithms to extract line segments have also been re-

F. Roli and S. Vitulano (Eds.): ICIAP 2005, LNCS 3617, pp. 220–228, 2005.

searched, but it is still difficult to detect line segments in real-time by one micro-processor.

In order to accelerate the LHT by hardware, many systems have been developed to date [1-11]. They can be categorized into three groups; parallel systems with many processing elements, dedicated hardware systems with ASICs, and reconfigurable systems with Field Programmable Gate Arrays (FPGAs). Of these three approaches, systems on reconfigurable devices are most promising at the present time because the performance of one reconfigurable device such as an FPGA is enough to realize real-time processing of the LHT, and the systems with reconfigurable devices can change their functions according to their circumstances.

In our system, (1) line segments (start and end points of lines) are detected (not only the lines by the LHT), and (2) more than 16348 line segments can be found in a standard size image in real-time, though it is far beyond the practical requirement, and not useful.

3 The Line Hough Transform

In this section, we overview the Line Hough Transform (LHT). In the LHT, a line is given by r (distance to the line from the origin) and θ (the angle of the line). In Figure 1(a), a line that goes through a point (x_i, y_i) is shown by the equation below.

$$r_k = x_i \times cos\theta_k + y_i \times sin\theta_k$$

In the LHT, when an edge point is given, all lines that go through the edge point are considered as candidates of the true line. In order to calculate (r, θ) of all the lines, the range of θ $(0 - \pi)$ is divided by N, and $(r_n, n/N \times \pi)$ $_{n=(0,N-1)}$ are calculated. With larger N, we can obtain more precise r and θ, but it requires more amount of memory and computation time. In Figure 1(b), three curves $(r = x_i cos\theta + y_i sin\theta)$ are plotted on (r, θ) plane, and each point on these curves shows a line that goes through (x_i, y_i). Among all points on these curves, the crossing points of the curves $((r_k, \theta_k))$ gives the true line.

In order to find the crossing points, for each edge point, N pairs of (r_i, θ_i) $(\theta = \pi \times i/N,\ i = 0, N-1)$ are calculated, and the value in (r, θ) memory are incremented using (r_i, θ_i) as addresses as shown in Figure 1(c). Then, peak values in (r, θ) memory are chosen as true lines in (x, y) plane.

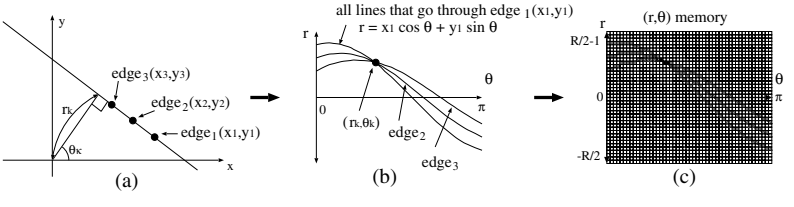


Fig. 1. The Outline of the LHT

4 Detection of Line Segments

In this section, we introduce the hardware algorithm to obtain line segments. In our approach, the following procedures are repeated twice to simplify memory access pattern and achieve higher performance.

1. Detect edge points in an given image.
2. Apply the Line Hough Transform (LHT) to the edge points.
3. Find start and end points of lines obtained by the LHT.

In the first path (*vertical search*), line segments in $\theta = [-\pi/4, \pi/4)$ are detected, while those in $\theta = [\pi/4, \pi \times 3/4)$ are detected in the second path (*horizontal search*). In our approach, $[-\pi/4, \pi \times 3/4)$ is used instead of $[0, \pi)$ as the range of θ. In *the vertical search*, data in the given image are read out along x axis (Figure 2(a)), while they are read out along y axis (Figure 2(b)) in *the horizontal search* in order to simplify non-maximum suppression in the edge detection, and discontinuity check in finding start and end points of line segments.

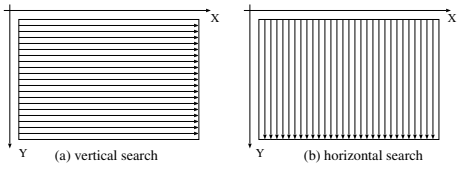


Fig. 2. The Scan Direction of Data in the Image

4.1 Edge Detection

First, the magnitude $M(x, y)$ of each pixel is calculated by Sobel filter.

$$M(x, y) = |G_h(x, y)| + |G_v(x, y)|$$

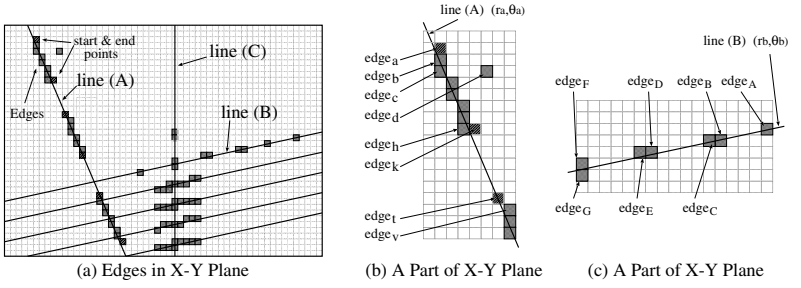
Then, non-maximum suppression is applied. In *the vertical search*, data in the image are read out along x axis as shown in Figure 2(a), and $M(x, y)$ is suppressed if $M(x, y)$ is smaller than $M(x - 1, y)$ or $M(x + 1, y)$. In *the horizontal search*, data are read out along y axis, and $M(x, y)$ is suppressed if $M(x, y)$ is smaller than $M(x, y - 1)$ or $M(x, y + 1)$. In our approach, orientation of an edge point is not calculated using G_v/G_h, but the search itself is divided by the orientation.

4.2 The Line Hough Transform

Suppose that the size of the image is $X \times Y$, and the range of θ is divided by N. Then, the computation order of the LHT becomes $X \times Y \times p \times N$, where p is the rate that a pixel in the image is an edge point. In this computation, the maximum parallelism can be N. The performance by parallel processing with N units and N memory banks is much faster than the video-rate in general. Therefore, parallel processing by N/m units is the best approach to achieve real-time processing with a smaller circuit.

4.3 Detection of Start and End Points of Lines

In Figure 3(a), line (A) goes on three line segments. In this case, only line (A) is detected by the LHT, and the three line segments can not be distinguished. Furthermore, line (B) and (C) that go through non-continuous edge points are also detected as lines because they make peaks in (r, θ) memory.



(a) Edges in X-Y Plane (b) A Part of X-Y Plane (c) A Part of X-Y Plane

Fig. 3. Lines Obtained by the LHT

Start and end points of lines can be found by checking pixels near the lines obtained by the LHT whether they are edge points or not. This approach is, however, not suitable for hardware systems because it requires non-regular memory accesses. We need a method to find start and end points of lines (1) which can be completely pipelined with the computation of the LHT, and (2) which does not require non-regular memory accesses.

A Procedure to Find out Start and End Points of Lines. In our method, four variables (*Start Point* (x, y), *Last Point* (x, y), *Counter* and *IsLine*) are given to each peak in (r, θ) memory. Using these variables, start and end points of lines are found as follows.

1. All *Counters* are initialized to zero.
2. For each edge point in the image, (r_i, θ_i) are calculated again.
3. Values in (r, θ) memory are read out using (r_i, θ_i) as addresses, and if (r_i, θ_i) is a peak,
   (a) If *Counter* is zero,
      i. the address of the edge point is set to *Start* and *Last Point*, and
      ii. *Counter* is incremented.
   (b) else (*Counter* is not zero)
      i. The distance between *Last Point* and the edge point is calculated.
      ii. *Counter* is incremented/decremented according to the distance, and
         A. If *Counter* becomes larger than a threshold, 1 is set to *IsLine*, and the threshold is set to *Counter*.
         B. If *Counter* becomes negative,
            – if *IsLine* is one, *Start* and *Last Point* are output as start and end points of a line,

– else, nothing is output.
Counter and *IsLine* are reset, and the address of the edge point is set to *Start Point*.

iii. The address of the edge point is set to *Last Point*.

4. After processing all edge points, (r, θ) memory is scanned, and if peaks whose *IsLine* are one are found, their *Start* and *Last Points* are output as start and end points of lines.

With this method, the computation to find start and end points can be completely pipelined with the computation of the LHT, because both computations use same input (same edge points) and finish in exactly same clock cycles.

In Figure 3(b), one of (r_i, θ_i) for $edge_a$ hits (r_a, θ_a), which is a peak in (r, θ) memory. Then, the address of $edge_a$ is set to its *Start* and *Last Point*, and *Counter* is incremented, because *Counter* was initialized to zero. In the computation of $edge_b$, *Counter* is incremented, because *Counter* is not zero, and the distance between *Last Point* $(edge_a)$ and $edge_b$ is zero. The address of $edge_b$ is set to *Last Point*. As for $edge_d$, no (r_i, θ_i) hits peaks in (r, θ) memory, and no variables on peaks are changed. By repeating the computation, *Counter* in (r_a, θ_a) becomes larger than the threshold, and its *IsLine* becomes one. When $edge_t$ is processed, the distance between *Last Point* $(edge_k)$ and $edge_t$ is large, and *Counter* becomes negative. Then, *Start Point* $(edge_a)$ and *Last Point*$(edge_k)$ are output as start and end points of a line. *Counter* and *IsLine* are reset, and $edge_t$ is set to *Start* and *Last Point*. In Figure 3(c), *Counter* in (r_b, θ_b) does not exceed the threshold, and nothing is output.

A Technique to Find Line Segments Under Noises. As shown in Figure 4(a), the actual edge points do not form an ideal line. Figure 4(b) shows a part of (r, θ) memory which are incremented by the edge points in Figure 4(a). In Figure 4(b), (r_k, θ_k) is chosen as a peak, but it is not sharp. In this case, $(r_i, \theta_i)_{\ i=0,N-1}$ calculated from the edge points do not always hit (r_k, θ_k) as shown in Figure 4(c). In Figure 4(c), only edge points in dark gray hit (r_k, θ_k), and other edge points hit $(r_k \pm l, \theta_k)_{\ l=1,\ldots}$. Therefore, we need a technique to consider $(r_k \pm l, \theta_k)_{\ l=0,\ldots}$ as one peak.

Figure 5 shows a technique to recognize $(r_k \pm l, \theta_k)_{\ l=0,\ldots}$ as one peak. In Figure 5, another memory (*R-Translation Table*) which has the same number of entries with (r, θ) memory is used, and when peaks in (r, θ) memory are found, address of *Start & End Points Table* is stored at $(r_k \pm l, \theta_k)$ in the *R-Translation*

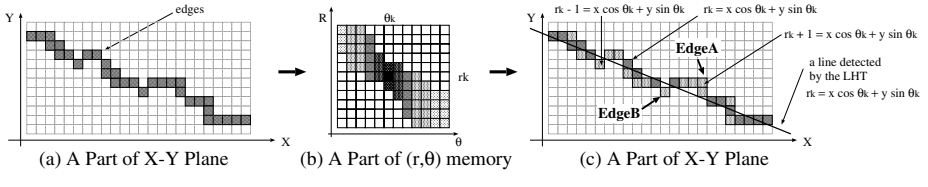


Fig. 4. Lines Obtained By Filtering

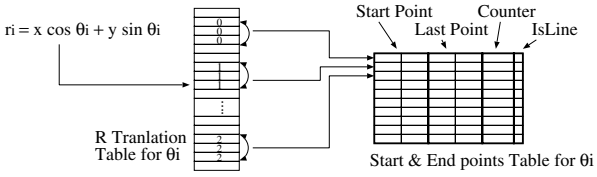


Fig. 5. R Translation Table

Table ($l = 0, 1$ in Figure 5). The address of *Start & End Points Table* starts from one (address zero means that it is not a peak), and incremented when a new peak is found in (r, θ) memory. When r_i for θ_i is calculated from an edge point, *R-Translation Table* is looked up, and if the value on (r_i, θ_i) is not zero, *Start & End Points Table* is accessed using the value as an address.

Distance Between Edge Points. In the technique described above, the scan direction of edge points is very important. In Figure 4(c), suppose that edge points are read out along X axis. Then, *EdgeB* hits (r_k, θ_k) next to *EdgeA*. These two edge points are not continuous, but we need to consider these points are continuous. This judgment becomes more difficult as θ of the line becomes closer to $\pi/2$, because the line becomes more parallel to X axis. Therefore, we need *the vertical and horizontal search* with different scan direction of edge points. In *the vertical search*, only y is used to decide the continuity of edge points, while only x is used in *the horizontal search*.

4.4 Parallel and Pipeline Processing

The sequence of the line segment detection described in the previous subsections can be summarized as follows.

1. Find edge points.
2. Calculate r_i, and increment values in (r, θ) memory.
3. Find peaks in (r, θ) memory.
4. Find start and end points of lines (start and end points of some line segments are output).
5. Output start and end points of lines scanning *Start & End Points Tables*.

In our implementation, N is 512. This means that θ is divided by 512, and 512 r are calculated for each edge point. By using smaller N, we can reduce computation time, but we can not get (r, θ) which matches well with long lines on the standard size image. The size of the (r, θ) memory becomes 512×800 ($N \times length\ of\ the\ diagonal$).

Figure 6 shows the parallel and pipeline processing realized in our circuit to achieve real-time visualization. In Figure 6,

1. addresses of edge points are generated by *Edge Detection Unit* and stored in an external memory.
2. the addresses of edge points are read out $(16 + 1) \times 2$ times.

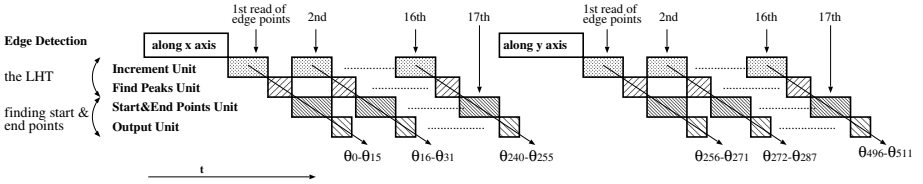


Fig. 6. Pipeline Processing

3. 16 θ_i are processed in each pipeline step with 8 units which run in parallel. Each unit processes two θ_i sequentially.
4. $(4+8+4)$ units run in parallel in the LHT in order to find peaks at least in the range of $(\theta_{i-4}, \theta_{i+4})$.
5. *Increment Unit* and *Start & End Point Unit* can not be active with *Find Peak Unit* and *Output Unit* at the same time, because they share same memory banks.

5 Performance

This circuit was implemented on an off-the-shelf PCI board (ADM-XRC-II by Alpha Data [13]) with one FPGA chip (Xilinx XC2V6000 [14]). The circuit runs at 66 MHz, and 45% of logic blocks, 33% of multipliers, and 29% of internal memory banks are used.

Table 1 shows the performance when the rate that a pixel in an image is an edge points is 5%, 7% and 9%. As shown in Table 1, the performance depends on the rate (the number of edge points in an image). The rate can be controlled by a threshold which is used to decide whether a pixel is an edge point in *Edge Detection Unit*. Lower threshold generates more edge points, but more noises are included. In this sense, the circuit can process more than 30 frames in one second under proper range of the threshold.

Figure 7 and 8 show the original image, and the edge points detected in *the vertical search*. Our algorithm is still sensitive to some parameters such as a threshold in the edge detection, the minimum peak height in the LHT, minimum length of line segments, and l (the parameter to recognize several (r, θ) as one peak described in 4.3). Figure 9 and 10 show the line segments detected by our circuit, when $l = [0, 1]$ and $[-1, 1]$ respectively. In Figure 9, all detected line segments fit well with the original image, but we can not detect long line segments because of the distortion of the camera and noises. In Figure 10, we can

Table 1. Performance of the Circuit

rate of edge points	clock cycles per image	frame per second
5%	1713290	38.9
7%	2131072	31.3
9%	2548864	26.2

Fig. 7. Original Image

Fig. 8. Edge Points (Vertical Search)

Fig. 9. Line Segment ($l = [0, 1]$)

Fig. 10. Line Segments ($l = [-1, 1]$)

detect long line segments, but some short lines do not fit well with the original image because of noises. We may need to control these parameters according to the length of line segments, though it requires more computation time.

6 Conclusions

In this paper, we described a compact circuit for real-time detection of line segments using the Line Hough Transform. This circuit was implemented on an off-the-shelf FPGA board with one FPGA chip. The circuit can find start and end points of line segments in an image (640×480 pixels) in real-time (more than 30 images per second). 45% of the FPGA chip is used for the circuit. With the rest of hardware resources, we can implement other circuits for higher level processing of object recognition, or improved the performance twice.

We are now improving the circuit to make it more robust to noises, and more insensitive to the parameters.

References

1. K. Hanahara, T. Maruyama and T. Uchiyama, "A real time processor for the Hough transform", IEEE Trans. Pattern. Anal. Mach. Intell. 10 (1987), pp. 121-125.
2. F.M. Rhodes, J.J. Disturi, G.H. Chapman, B.E. Emerson, A.M. Soares and J.I. Raffel, "A monolithic Hough transform processor based on restructurable VLSI", IEEE Trans. Pattern. Anal. Mach. Intell. 10 (1988), pp. 106-110.
3. M.F.X.D Van Swaaij, F. V. M. Catthoor and H. J. De Man, "Deriving ASIC architecture for the Hough Transform", Parallel Computing 16, 1990, pp.113-121
4. M.Atiquzzamau, "Pipelined implementation of the multi-resolution Hough Transform in a pyramid multiprocessors", Pattern Recognition Letters, 1994, pp.841-851.
5. D. Ben-Tzvi, A. Naqui, M. Sandler, "Synchronous multiprocessor implementation of the Hough Transform", Computer Vision Graphics Image Process 1990, pp.437-446.
6. A-N. Choudhary and R.Ponnussary, "Implementation and evaluation of Hough transform algorithm on shared-memory multiprocessors", J. Parallel Distributed Comput. 12, 1991, pp.178-188.
7. A. L. Abbott, P. M. Athanas, L. Chen, and R. L. Elliott, "Finding Lines and Building Pyramids with Splash 2", FCCM 1994.
8. Chung, K.L., Lin, H.Y., "Hough Transform On Reconfigurable Meshes", Computer Vision and Image Processing, No. 2, March 1995, pp. 278-284.
9. M. Nakanishi and T. Ogura, "Real-time line extraction using a highly parallel Hough transform board", Proceedings of International Conference on Image Processing, 1997, pp. 582-585.
10. Pan, Y., Li, K., Hamdi, M., "An Improved Constant-Time Algorithm for Computing the Radon and Hough Transforms on a Reconfigurable Mesh", IEEE Trans. Systems, Man and Cybernetics-A(29), No. 4, July 1999, pp. 417.
11. Tagzout, S., Achour, K., Djekoune, O., "Hough transform algorithm for FPGA implementation", Signal Processing 81, No. 6, June 2001, pp. 1295-1301.
12. N. Nagata and T. Maruyama, "Real-time Detection of Line Segments Using The Line Hough Transform", IEEE International Conference on Filed-Programmable Technology, 2004, pp. 89 - 96.
13. http://www.alpha-data.com
14. http://www.xilinx.com