HARDWARE IMPLEMENTATION OF A 2D MOTION ESTIMATION SYSTEM BASED ON THE HOUGH TRANSFORM

*Hsiang-Ling Li and Chaitali Chakrabarti**

Department of Electrical Engineering
Telecommunications Research Center
Arizona State University
Tempe, Az 85287-5706
li@dspsun.eas.asu.edu, chaitali@asu.edu

ABSTRACT

A novel feature-domain 2D motion estimation system based on the straight-line Hough transform (SLHT) is presented. It operates on 256×256-pixel binary images and consists of two main blocks. The first block does preprocessing work including smoothing the boundary, tracing and integrating the contours, and detecting dominant points. The second block computes the Hough transform on contour segments as well as the rotation and translation parameters. Each of the modules has been implemented (gate level) and simulated using Mentor Graphics tools.

1. INTRODUCTION

Motion estimation is a key issue in dynamic scene analysis problems with applications in a wide range of fields including video coding, data compression, object tracking and autonomous navigation. Most of the past work in motion estimation of 2-D objects have been based in the pixel-domain, and require a large number of computations. Feature-domain based approaches, on the other hand, represent an object by a small set of features (such as shape features, moments, Fourier transforms), resulting in a reduction in the number of computations.

The Hough transform (HT) is a well-known method for the detection of parametric curves and has been used to estimate the amount of translation and rotation of 2D objects in [1]. The objects were, however, limited to analytic curves such as circles, ellipses, and parabolas. A more recent work [2] has generalized and extended this method for arbitary shapes by utilizing the perpendicular distances between pairs of parallel tangent lines as the signature of a shape.

Some hardware implementations of HT-based systems have also been reported in the literature [3]-[5]. In [3], a vision based vehicle guidence system is developed, which has applied the algorithms of gradient filtering, binarization, edge thinning, line verification and HT to extract two most distinct line representing the street boundaries. A chip-set of a real time system based on Generalized Hough transform for object recognition of bi-dimensional images is described in [4].

*This work was supported by a grant from NSF, no. MIP-93095.

In this paper, we present a novel feature-domain 2D motion estimation system based on the straight-line Hough transform (SLHT). The system is based on the motion estimation technique of [6] that represents objects uniquely by straight-line approximations of the boundary $\{(\theta, p)\}$, and estimates the motion parameters from shifts in the $\theta - p$ space. The straight-line approximation of the boundary is obtained by detecting dominant points (or high curvature points) during contour tracing, and then approximating the pixels between adjacent dominant points (along the contour) by a straight line using the Hough Transform. The system has been implemented using Mentor Graphics tools. It operates on 256×256-pixel binary images and consists of two main blocks. The first block traces the contours, and detects dominant points while the second block computes the Hough transform on contour segments. The second block also computes the rotation and translation parameters.

2. ALGORITHM SUMMARY

In order to reduce the number of computations in motion estimation, the boundary of an object needs to be represented compactly. In the existing procedures [1], [2], the boundary of an object is approximated by tangent lines through almost every boundary pixel. Such a procedure requires an enormous amount of computation. Moreover, a threshold value in the $\theta - p$ space is required to reduce the number of spurious peaks. In [6], we describe a method to represent an object compactly using a *small* set of straight-lines, where each straight-line approximates the contour between two consecutive dominant points. We use dominant points in the representation of an object since it is well known that information on a 2-D shape is concentrated at the dominant points [7].

The first step in our procedure is to trace the contour and detect the dominant points. Next, the contour is broken into groups of edge pixels with dominant points serving as the end points of each group. Then, each group of edge pixels is approximated by a straight line. The reason why we do not rely solely on dominant points to represent the object is that discretization errors can cause displacements in the positions of the dominant points. This is especially true when objects are rotated. By computing the Hough transform of pixels between two adjacent (along the contour) dominant points, an object is represented more robustly. No thresholding is required here, since each group of pixels gives rise to a single peak for one straight line.

Once an object is represented uniquely by straight-line approximation of the boundary or by a set of (θ, p)s, we estimate the motion parameters of the object by computing the shifts in the $\theta - p$ space. Let $\{(\theta, p)\}_{t_a}$ and $\{(\hat{\theta}, \hat{p})\}_{t_b}$ be the representation of the object at times t_a (reference object) and t_b (test object) respectively. We first determine the rotation angle ϕ by computing the votes on $\Delta\theta$, the difference in the θ values of the object at times t_a and t_b. Then we compute the approximate translation vector from the displacement of the centroid of the polygon whose vertices are the dominant points. We then refine the approximate translation vector by computing the inverse Hough Transform on $\{(\theta, \Delta p)\}$ where $\Delta p = \hat{p}(\hat{\theta}) - p_0(\theta)$, and $p_0(\theta)$ is the intrinsic term obtained from $p(\theta)$ of the reference object. This procedure has been implemented in software. The results have been accurate and have been presented in [6].

3. IMPLEMENTATION OF THE SYSTEM

The motion estimation system consists of two main blocks: the preprocessor and the HT processor. Both the blocks have been designed (gate level) and simulated using the Design Architect tool and the Quicksim tool of Mentor Graphics.

3.1. Preprocessor

The block diagram of the preprocessor is shown in Figure 1. The noisy boundary pixels of the input image are removed by the "boundary smoothing module". Then, the "contour tracer" detects boundary pixels and the "contour integrator" links the broken contours. Last, the dominant points are detected by the "dominant point detector", and sent to the HT processor.

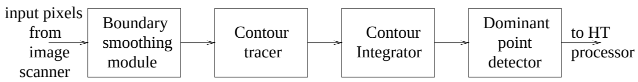

Figure 1. Block diagram of the preprocessor.

Boundary smoothing module: This unit is implemented by a state machine, which can be easily implemented using a NAND decoder. The state machine has 40 states with every state corresponding to a certain combination of 9 pixels (3×3 window). Whenever a combination of 9 pixels matches with any one of the 40 states, the corresponding output line goes high, i.e. a state is detected. Since we use a 3×3 window and work directly on the raster input pixels to smooth the boundary, two line delays are required. The two line delays are implemented using two RAMs, shift registers and transparent latches. Each RAM stores 253 pixels or bits with every stored bit accessed by some address.

Contour tracer: In this estimation system, the image frame is partitioned into 16 subimages, and the contours in each subimage (64×64 pixels) are generated in parallel. Next, contour integration is done to link the contours in each subimage. The contour tracer consists of 16 processors with each processor assigned to a subimage for contour tracing and one postprocessor (contour integrator) to do contour integration. The architecture of the contour tracer is shown in Figure 2.

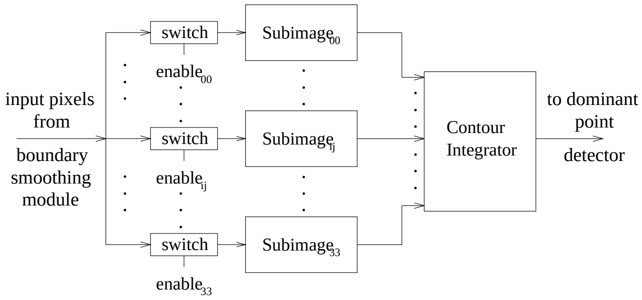

Figure 2. Contour tracer architecture.

Each processor stores the pixels assigned to it by the input switches in the "subimage memory". Each subimage processor has a state machine (NAND decoder) with 28 states and is responsible for tracing the contours of a subimage. First, the processor detects the starting pixels of all the contours and stores the corresponding $x - y$ coordinates in the "starting point memory". Then, it starts tracing the contour from one of the starting pixels. During contour tracing, if a match occurs on any of the 28 states, the next edge pixel (which is the center pixel of the next 3×3 window), and the chain code of the current edge pixel relative to the previous edge pixel (in the contour) are both determined and stored in the "contour memory". Once the boundary of a subimage is reached, the end pixel of a contour is detected and stored in the "end point memory". The address of an end point (end index) in the contour memory is also stored in the "end index memory". The contour tracing procedure then starts from the next starting pixel.

Contour integrator: The architecture of the contour integrator is shown in Figure 3. Each contour segment is represented by two values, the start index (equal to the previous end index added by one) and the end index. Both indices are the addresses in the contour memory for one subimage. First, all the starting pixels of each subimage, A_1, B_1, and so on are loaded into memory 1. All the subimage indices $((i, j))$, start indices, end indices and related end pixels, A_2, B_2, and so on are loaded into memory 2. Then, this unit links two contours by matching the starting pixel of one contour with the end pixel of another contour. All the broken contours are sorted and stored in the "sorted contours memory" (memory 3). Last, the contour pixels and chain codes from the start index to the end index are loaded from the contour memory in a subimage to the "final contour memory". Once the last contour segment is loaded into the final contour memory, the task of contour integration is completed.

Dominant point detector: This unit consists of two parts. The first part detects the region of support, and has

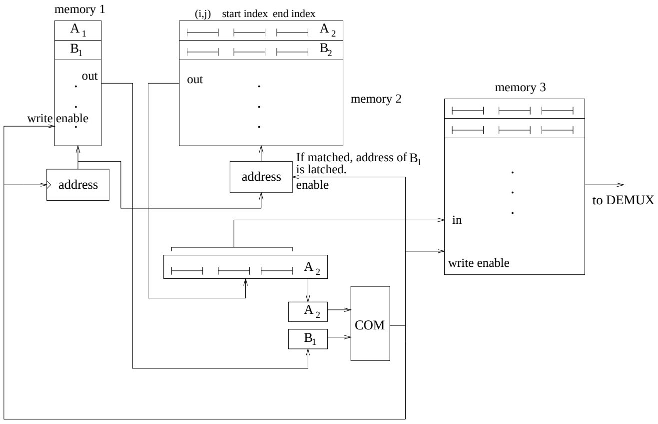

Figure 3. Contour integrator.

8 cells corresponding to the 8 chain codes. Each cell has a counter and related logic to determine if the number of consecutive pixels with the same chain code can contribute to a large corner or a small corner (a convex/concave region). Two pointers (addresses in the "final contour memory") are generated: front pointer and rear pointer, which specify a region of support. The second part computes the curvature of each pixel within a region of support using the k-curvature method and uses the curvature of the middle point in the region as the reference value. The pixel having the maximal curvature is the dominant point.

3.2. Hough Transform (HT) Processor

The contour segment (bound by dominant points) are sent to the HT-processor one after the other. The HT-processor then outputs a (θ, p) value for each contour segment. The block diagram of the HT-processor is shown in Figure 4. Consider a contour segment: (x_0, y_0), (x_1, y_1), ..., (x_n, y_n) (loaded from the "final contour memory"), where (x_0, y_0) and (x_n, y_n) are the dominant points. The first and second blocks use the two end points to compute $\tilde{\theta}$, the initial estimate of θ, $\tilde{\theta} = tan^{-1}(-\frac{x_n - x_0}{y_n - y_0})$. The Hough transform of (x_0, y_0) is computed around $\tilde{\theta}$ in the third block. The last block computes the p values of other pixels by simple addition or subtraction and accumulates votes to estimate (θ, p) for the contour segment.

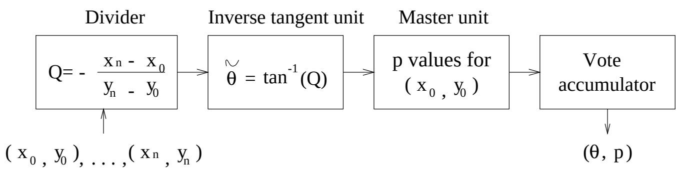

Figure 4. Block diagram of the HT-processor.

Divider: The quotient $Q = -\frac{x_n - x_0}{y_n - y_0}$ is computed using CORDIC arithmetic. When $Q \geq 8$, $\tilde{\theta}$ is close to 90^o. Hence it is good enough for initial estimates to use the following equation with 8 iterations to compute Q, $Q_{m+1} = Q_m + (y_n - y_0) * \delta_m * 2^{2-m}$, $m \leq 7$ and $Q_0 = x_n - x_0$. δ_m is determined by the MSBs of Q_m and $(y_n - y_0)$ (using 2's complements as negative numbers). Then, $Q = -\frac{x_n - x_0}{y_n - y_0} \cong \sum_{m=0}^{7} \delta_m * 2^{2-m}$. The architecture of the divider is shown in Figure 5. The δ_m is generated by a two-input XNOR gate.

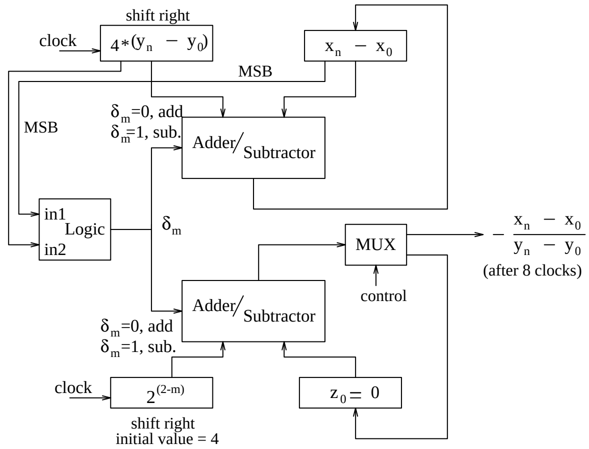

Figure 5. Architecture of the divider.

Inverse tangent unit: For $Q \geq 1$, we use a look up table to determine $\tilde{\theta}$. For instance, if $Q \geq 8$, then $\tilde{\theta} = 83^o$. However, if $Q < 8$ but ≥ 4, then $\tilde{\theta} = 76^o$. In such cases, the following equation, $\tilde{\theta} = \sum_{j=0}^{7} \delta_j * \theta_j$, where $\theta_j = tan^{-1}2^{-j}$, is solved to determine δ_js. For $Q < 1$, we can directly solve the following equation, $Q \cong \sum_{j=0}^{7} \delta_j * 2^{-j}$ to determine δ_js. The δ_js are sent to the master unit.

Master unit: This unit consists of two identical subunits with each subunit implementing the following two equations: $p_{j+1} = p_j + \delta_j * 2^{-j} * q_j$ and $q_{j+1} = q_j - \delta_j * 2^{-j} * p_j$, $j = 0, \ldots, 7$. These two subunits work in parallel to save the overall computation time and reduce storing complexity. The first subunit computes the p value at $\tilde{\theta}$ for (x_0, y_0), i.e. $p_0(\tilde{\theta})$ by using $p_0 = x_0$ and $q_0 = y_0$. The second subunit computes $sin\tilde{\theta}$, and $cos\tilde{\theta}$ in parallel by using $p_0 = 1$ and $q_0 = 0$. It takes 8 clock cycles to compute these values. "K_n", a machine constant is not included in the computation since it does not effect the final voting result. Let the quantization level $\Delta\theta$ be such that $tan^{-1}(\Delta\theta) = 2^{-6}$, i.e. $\Delta\theta \cong 0.9^o$. Then, the other p values for (x_0, y_0) at $\tilde{\theta} \pm \Delta\theta$, $\tilde{\theta} \pm 2\Delta\theta$,..., and $sin(\tilde{\theta} \pm \Delta\theta)$, $cos(\tilde{\theta} \pm \Delta\theta)$, $sin(\tilde{\theta} \pm 2\Delta\theta)$, $cos(\tilde{\theta} \pm 2\Delta\theta)$ ... can be easily computed in the first and second subunits. Each of these values is computed in one clock cycle. These p values as well as the sine and cosine of the corresponding angles are transmitted to the vote accumulator and used as initial values to compute p values for other pixels in the contour segment. The architecture of the subunit is shown in Figure 6.

Vote accumulator: This unit consists of a linear array of cells where each cell computes the p values corresponding to some θ value for all the pixels in the contour. If the range of θ is $\tilde{\theta} - k * \Delta\theta$ to $\tilde{\theta} + k * \Delta\theta$, then the linear array consists of $(2k+1)$ cells. Each cell stores $p_0(\theta)$, $sin\theta$, and $cos\theta$ and computes the p value of the jth pixel, p_j, from the p value of the $(j-1)$th pixel, p_{j-1}, by $p_j = p_{j-1} + b_{j1}cos\theta + b_{j2}sin\theta$, where θ is the angle assigned to that cell, and b_{j1}, b_{j2} are determined from the difference in the $x - y$ coordinates of

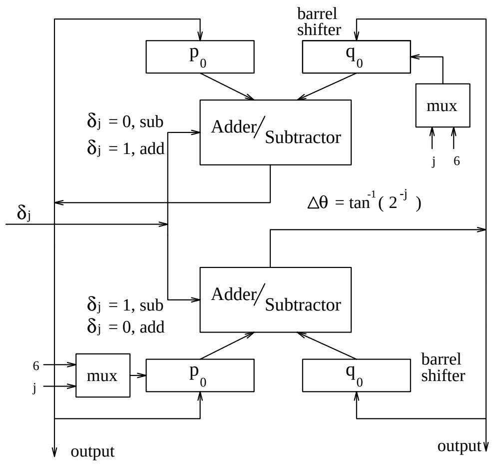

Figure 6. Architecture of the subunit in the HT-processor.

the jth and $(j-1)$th pixel. Once a p value is computed, it is latched as the address to the "vote memory" which stores the votes, and the corresponding vote is incremented and stored back in the same address. In each evaluation, the updated vote is compared to the maximum stored vote (initial value=0) and the larger vote is stored as the maximum vote. The coresponding p value is also stored. In the end, each cell detects the largest vote and its corresponding p. A comparison of these values among the cells is used to to obtain the (θ, p) for one contour segment. The architecture of the vote accumulator is shown in Figure 7.

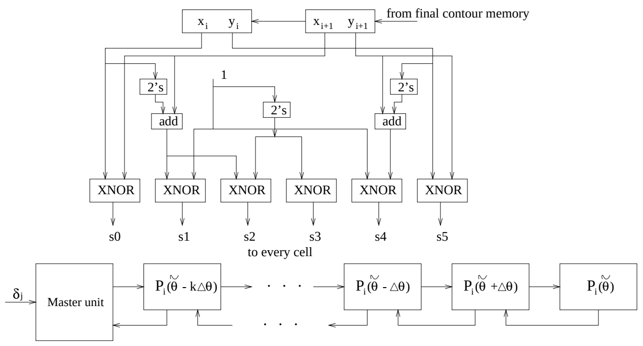

Figure 7. Architecture of the vote accumulator.

Estimation of the rotation angle: The vote accumulator can be used to compute the votes on $\Delta\theta$ to obtain the initial estimate of the rotation angle, $\tilde{\phi}$. Since the differences between any two corresponding angles θ and $\hat{\theta}$ are not exactly equal [6], the detected peak does not always correspond to the correct rotation angle. Let $\{(\theta, p)\}_{t_a}$ be stored in the "reference image memory" and $\{(\hat{\theta}, \hat{p})\}_{t_b}$, be stored in the "test image memory". By matching $\hat{\theta}_j$ with $\theta_i + \tilde{\phi} \pm \Delta\tilde{\phi}$, a one to one correspondence can be obtained between matched $\{(\theta, p)\}_{t_a}$ (denoted as $\{(\theta, p)\}'_{t_a}$) and $\{(\hat{\theta}, \hat{p})\}_{t_b}$ (denoted as $\{(\hat{\theta}, \hat{p})\}'_{t_b}$). Then, the differences between the corresponding angles θ (in $\{(\theta, p)\}'_{t_a}$) and $\hat{\theta}$ (in $\{(\hat{\theta}, \hat{p})\}'_{t_b}$) are averaged for a better estimate of the rotation angle, ϕ.

Estimation of the translation vector: The second subunit (in the master unit of the HT processor) computes the translation term, $T(\theta_i) = t_0 cos(\theta_i - \alpha_0)$ by using $p_0 = t_0$ and $q_0 = 0$ ($i = 1, \ldots, n$ and n is the number of dominant points). The intrinsic term, $p_0(\theta_i)$, is calculated for each θ by subtracting the corresponding translation term, $T(\theta_i)$ from $p(\theta_i)$ of the reference object. Then, $\Delta p(\theta_i) = \hat{p}(\hat{\theta}_i) - p_0(\theta_i)$ is calculated. The divider computes the slopes m_is, where $m_i = -cos\hat{\theta}_i / sin\hat{\theta}_i$, and $y-$intercepts b_is, where $b_i = \Delta p_i / sin\hat{\theta}_i$. The vote accumulator then computes the votes in the $x-y$ plane (inverse HT) to obtain (c'_x, c'_y), the center point of the object at time t_b. Finally, the translation vector $(c'_x - c_x, c'_y - c_y)$ is obtained using a simple subtractor, where (c_x, c_y) are the coordinates of the centroid of the polygonal approximation of the obect at time t_a.

4. CONCLUSION

The HT based motion estimation algorithm has been mapped onto hardware. The motion estimation system operates on 256 × 256-pixel binary images and consists of two main blocks. The first block does preprocessing work (boundary smoothing, contour tracing, and dominant point detection). The second block computes the Hough transform on contour segments as well as the rotation and translation parameters. Each of the modules has been implemented (gate level) and simulated using Mentor Graphics tools. The entire system is currently being simulated.

REFERENCES

[1] D.Casasent and R.Krishnapuram, "Curved object location by Hough transformations and inversions," *Patt. Recogn.*, vol. 20, no. 2, pp. 181-188, 1987.

[2] D.C.W.Pao, H.F.Li, and R.Jayakumar, "Shapes recognition using the straight line Hough transform: theory and generalization," *IEEE Trans. on PAMI*, vol. 14, no. 11, pp. 1076-1089, Nov. 1992.

[3] J.Schonfeld and P.Pirsch, "Image processing board for real-time extraction of line symbols from video sequences," VLSI Signal Processing VI, 1993, pp. 30-38.

[4] A.Antola et. al., "A chip-set for the Hough transform", *VLSI Signal Processing VI*, pp. 129-138, 1993.

[5] K.Hanahara et. al., "A real time processor for the Hough transform", *IEEE Trans. PAMI*, vol. 10, pp. 121-125, 1988.

[6] H.L.Li and C.Chakrabarti, "Motion estimation of two-dimensional objects based on the straight-line Hough transform: A new approach," accepted for publication in *Pattern Recognition.*

[7] C.H.Teh and R.T.Chin, "On the detection of dominant points on digital curves," *IEEE Transactions on PAMI*, vol. 11, no. 8, pp. 859-872, 1989.

HARDWARE IMPLEMENTATION OF A 2D MOTION ESTIMATION SYSTEM BASED ON THE HOUGH TRANSFORM

Hsiang-Ling Li and Chaitali Chakrabarti[1]

Department of Electrical Engineering
Telecommunications Research Center
Arizona State University
Tempe, Az 85287-5706
li@dspsun.eas.asu.edu, chaitali@asu.edu

A novel feature-domain 2D motion estimation system based on the straight-line Hough transform (SLHT) is presented. It operates on 256×256-pixel binary images and consists of two main blocks. The first block does preprocessing work including smoothing the boundary, tracing and integrating the contours, and detecting dominant points. The second block computes the Hough transform on contour segments as well as the rotation and translation parameters. Each of the modules has been implemented (gate level) and simulated using Mentor Graphics tools.