Local Lines: A Linear Time Line Detector

Sharlee Climer
Department of Computer Science
Washington University
St. Louis, MO 63130-4899

Sanjiv K. Bhatia
Department of Computer Science
Southern Illinois University Edwardsville
Edwardsville, IL 62026-1656

Abstract

This paper introduces Local Lines – a robust, high-resolution line detector that operates in linear time. Local Lines tolerates noisy images well and can be optimized for various specialized applications by adjusting the values of configurable parameters, such as mask values and mask size. As described in this paper, the resolution of Local Lines is the maximum that can be justified for pixelized data. Despite this high resolution, Local Lines is of linear asymptotic complexity in terms of number of pixels in an image. This paper also provides a comparison of Local Lines with the prevalent Hough Transform Line Detector.

Keywords: Line detector, Linear complexity, Low level feature extraction

1 Introduction

Lines are fundamental low level features in images. Line detection is important for a wide variety of applications in computer vision and digital image processing. For example, line detectors facilitate applications such as recognition, navigation, camera calibration, and target tracking [2, 4, 8]. A typical line detector operates on an *edge image*. An edge image can be derived from a given image by using an edge detector or it can be created by plotting sensor data. Typically, an edge image is a binary image with a value of 1 at each edge point location, and zeros at all other locations. The binary image is used by a line detector to determine the defining characteristics of lines found in the image, such as slope, length, midpoint, and/or endpoints of the segment.

The importance of line detection has led to the development of a number of algorithms for this operation but most of them are computationally expensive [2, 3]. The Hough Transform Line Detector (HTLD) is a prevalent algorithm in use [6]. For this reason, we use the HTLD as a basis for our comparisons. In this section, we describe the HTLD, briefly introduce Local Lines, and outline the organization of this paper.

1.1 Hough Transform Line Detector

The HTLD draws on global properties of images to derive global properties of probable lines [1]. Examples of global properties of lines are slope and y-intercept. In contrast, local properties of lines refer to specific information about line segments such as length, midpoint, and endpoints, as well as slope. HTLD is used to identify probable lines that span the entire image and then the results are postprocessed to identify the endpoints of the actual line segments. The spanning lines are identified using two parameters that are necessary and sufficient for the purpose. When Paul Hough introduced this algorithm in 1962, he specified the slope and y-intercept as these parameters [5]. This selection led to difficulties in practice, as it results in an infinite parameter space [8]. Since that time, a number of parameter pairs have been suggested. Two popular choices are the ρ and θ pair and the Muff Transform. Duda and Hart suggested ρ and θ, where ρ is the length of a normal of the spanning line to the origin and θ is the angle this normal makes with the positive x-axis [5]. Figure 1 depicts an example spanning line with the ρ and θ parameters identified.

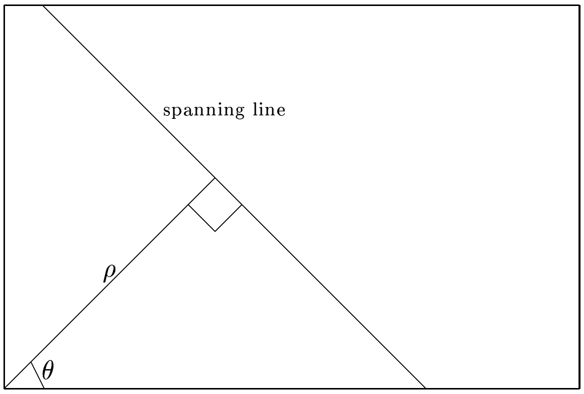

Figure 1: The use of ρ and θ as parameters

As shown in Figure 2, the Muff Transform uses the two points where the spanning line intersects the edges of the image [5]. The transform starts at the lower left corner of the image and measures in a counter-clockwise direction along the perimeter of the image. It defines two parameters – s_1 and s_2 – as the distance to the first and second intersection points, respectively.

Parameter pair selection is important as the possible range of each parameter must be quantized, and the speed and quality of the HTLD is dependent on this quantization. An accumulator cell is initialized for each possible pair of discrete

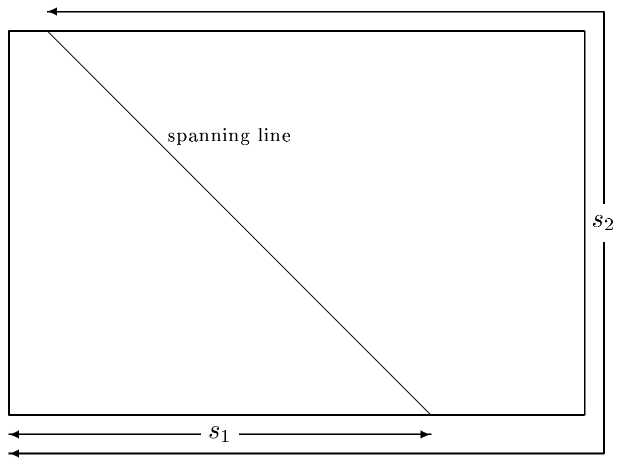

Figure 2: The Muff Transform parameter pair

parameter values. Thus, both space and time requirements of the HTLD increase rapidly with increasing numbers of discrete values. Unfortunately, reducing the number of intervals also reduces the resolution of the detector.

The algorithm for the HTLD, as described in [8] and using the $\langle \rho, \theta \rangle$ parameter space, is presented in Figure 3. The input is an $M \times N$ edge image. The range of ρ is $[0, \sqrt{M^2 + N^2})$ and the range of θ is $[0, \pi)$ [8]. R and T are the number of discretized values for ρ and θ, respectively. These values should be chosen to allow maximum resolution, given time constraints.

The algorithm proceeds as follows. An $R \times T$ matrix A of accumulator cells is initialized to zero. The edge image is searched row by row to find each edge point. When an edge point is encountered, T calculations are performed. For each quantized value of θ, calculate ρ for the given edge point. Increment the corresponding accumulator cells for each calculation. Thus, for each edge pixel, T accumulator cells are incremented, one for each discrete value of θ.

After all edge points have been explored, accumulator cells corresponding to spanning lines containing large numbers of edge points have higher tallies than those with lower numbers of edge points. Therefore, probable lines are specified by accumulator cells with tallies greater than a pre-specified threshold. Challenges associated with deriving an optimal threshold and with reducing peak splitting among adjacent parameter pairs are detailed in [5].

HTLD indicates the lines by their global properties, *i.e.*, their parameter pair values. However, the parameter pair does not contain information about the actual location of the line segment, or whether a detected spanning line contains a single, long line segment or several short line segments. Furthermore, the tally for a parameter pair includes every edge pixel that happens to occur along the spanning line, even if it is not part of any line segment at all. This shortcoming may result in inappropriately high tallies for some parameter pairs.

```
Input: An M x N image E, with E[i][j] = 1 for edge point,
         0 otherwise.
Arrays rho[0..R] and theta[0..T] containing discretized values
         of rho and theta.
A[]: An R x T matrix of accumulator cells, initially set to zero.

algorithm houghlines ( image E )
    for each pixel E[i][j]
        if ( E[i][j] == 1 )
            for ( h = 1...T )
                x = i cos ( theta[h] ) + j sin ( theta[h] )
                determine the index, k, of rho[] closest to x
                increment A[k][h]
    for ( k = 1...R)
        for ( h = 1...T)
            if ( A[k][h] > threshold )
                print rho[k], theta[h]
```

Figure 3: HTLD algorithm

Moreover, it could result in false-positive lines being detected. For example, if a small textured or patterned object occurs in an image, there would be a high concentration of edge pixels in the immediate area. These edge pixels would increase the tallies of the corresponding spanning lines, although there is no real line segment present. For these reasons, the image and the output need to be postprocessed to find what lines actually exist, and the endpoints and lengths of these lines.

The complexity of the HTLD is a function of the number of pixels in the image as well as the number of intervals, R and T, chosen for the parameters. In order to get satisfactory results, larger images would require greater numbers of intervals. We compare the complexity of HTLD and Local Lines in Section 3.

1.2 Local Lines

A drawback of HTLD is that it uses global properties of images, thereby increasing the complexity of the program. It returns global information about each of the detected lines which requires additional processing to determine the actual locations and lengths of the lines.

With this in mind, we have designed and implemented a line detector, Local Lines, that draws on local information and outputs local properties of each probable line.

In Local Lines, we apply a mask on each edge point, to yield weighted sums of other local edge points and return probable n-pixel long line segments. These segments are then deleted and a tag is placed in the far endpoint to facilitate the catenation of segments. This process is linear in the number of pixels, so its complexity is dramatically less than the HTLD and it scales well. Furthermore, the program returns the endpoints, slope, and length of each line, ready for use without further processing.

The rest of the paper is organized as follows. In Section 2, the Local Lines algorithm is described in greater detail. In Section 3, we derive its resolution and complexity and compare the results with the HTLD. Section 4 describes our implementation of this algorithm and illustrates the output of Local Lines. Finally, Section 5 summarizes our conclusions and ideas for future work.

2 Local Lines Algorithm

The algorithm for Local Lines is summarized in Figure 4. As with HTLD, the input to the system is an edge image. The edge image is scanned one row at a time until an edge point is encountered. When an edge point (i, j) is encountered, a convolution mask is applied to the system. In our system, the convolution mask for an n-pixel line is $(n + 2) \times (2n + 1)$, where n is the minimum length of identified lines. In our experiments, we set n equal to ten pixels, resulting in a 12×21 mask (see Figure 5). The mask is aligned such that the pixel (i, j) is under the second row and middle column of the mask. We are interested in finding line segments that start at (i, j) and end below row i or horizontally to the right of pixel (i, j). The numbers along the edges of the mask in Figure 5 correspond to possible endpoints for these segments. Each of these endpoints are labeled from 0 to $4n - 1$ (0 to 39 in our experiments for 10-pixel lines). This labeled number is referred to as the *slope* throughout the program, with the slope converted to a standard format in the results. We have a counter for each slope, and we initialize these slope counters to zero each time we apply the mask.

The next step is to find all the local edge points in the image under the convolution mask. For each of these other local edge points, the counters corresponding to the possible slopes of lines that could contain both that point and the point (i, j) are incremented by a weighted number. The weight of the number reflects the likelihood that the point lies on a line that starts at point (i, j) and has the corresponding slope. The method used to determine this weight is demonstrated for the slope of 0 in Figure 6. A two-pixel wide rectangle is centered on the point (i, j) and a given endpoint along the sides or bottom edge of the mask. Then the aligned rectangle is shaded. The shaded area of each pixel is used for the weight of the corresponding pixel for the given slope. For example, for slope 0, the mask weights are one for pixels located horizontally to the right of (i, j) and one-half for pixels directly above or below these horizontal pixels.

The selection of this weighting method is somewhat arbitrary. Experiments with alternative weighting methods, as well as different mask sizes and/or shapes, may result in enhanced system performance. Note that once a mask size, shape, and weight values have been determined, the values can be stored in a file. Thus, the values need not be calculated during the application of the

```
Input is an M x N image E, with E[i][j] = 1 for edge point,
      0 otherwise

algorithm local_lines ( image E )
{
  for each pixel E[i][j]
    if ( E[i][j] == 1 )
      apply slope mask and update counters
      for each counter
        if ( counter > tolerance )
          check for fan of lines
          if E[i][j] or neighbor is endpt. & slopes are similar
            catenate lines
          else
            start a new line
          delete pixels in line
          tag endpoint with line number
      reset counters
}
```

Figure 4: Local Lines algorithm

algorithm; they only need to be recomputed to fine-tune the code for a particular application domain.

We used a 3-dimensional array (shown in Figure 7) to implement the weighted mask. The rows and columns in this 3D array correspond to the convolution mask (12×21 mask in our example). The third dimension corresponds to the possible slopes for an n-pixel line. There are $4n$ of these possible slopes. Thus, for a 10-pixel line, the 3D mask has dimensions of $12 \times 21 \times 40$. This 3D mask facilitates the computation of values for a given edge pixel. For each edge point found under the 12×21 convolution mask projection, the weighted values along the third dimension (the slope axis) are added to the corresponding slope counters.

Once all of the edge points have been found under the 2D mask area, the slope tallies are checked for totals greater than or equal to the tolerance level, as specified by a configurable parameter `min_points`. The parameter `min_points` can be adjusted to tailor the system to the quality of the input. Thus, the system is robust and tolerates missing and contaminated data well. The selected values correspond to probable n-pixel line segments starting at the point (i, j), with the given slope.

It is possible to have a *fan* of line segments, that is, several line segments with adjacent slopes values. For instance, if all the edge pixels of a given line were detected, using the weights of the current mask and `min_points` equal to a small number, several lines may be detected with adjacent slope values. Since

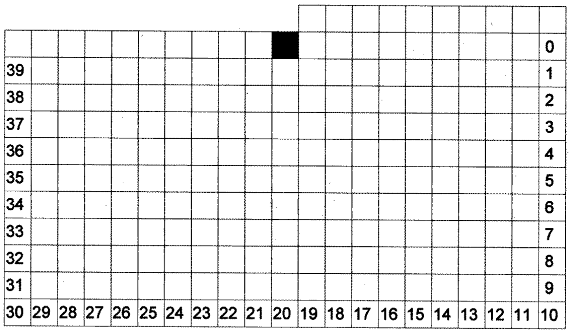

Figure 5: 12 × 21 mask

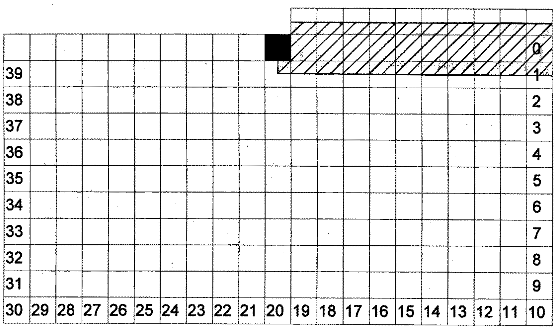

Figure 6: Two-pixel wide rectangle used to determine weights for pixels for a line with slope 0

we can assume that there would not be two or more lines with the specified start point and very similar slopes, we should single out one line from the fan. Local Lines does this by comparing the tallies of each line in the fan and selecting the slope with the highest tally.

Once a line is determined, its properties are stored and the pixels in the line are deleted. The endpoints and slope of the line are stored in an array, with each element in the array representing a line. The index of the line is placed at the segment's endpoint on the image, to facilitate catenation with other line segments. For this reason, we assign the first line that is encountered, to have an index 2. This ensures that the line segment markers are not confused with edge points. Other pixels in the line are deleted from the image by setting them to zero. The corresponding slope mask is used to select the pixels to be deleted. If the weight for the pixel is greater than a threshold value specified by a configurable parameter `deltol`, then the pixel is deleted.

The new line may be a continuation of an existing line segment. If the pixel

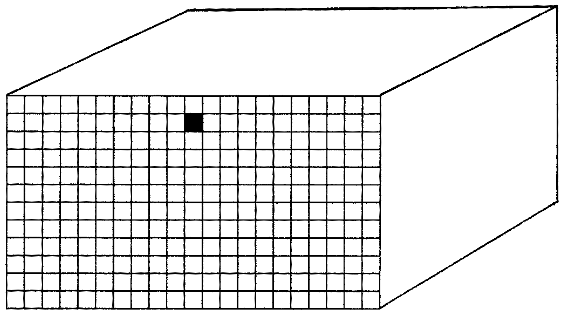

Figure 7: 3-dimensional mask matrix

at or in the neighborhood of location (i, j) has a value greater than one, it is the endpoint of another line whose index is the value of the pixel. In this case, the slopes of the new line and nearby existing lines are compared. If they are within a given tolerance, specified by a configurable parameter `slopetol`, then the new line is catenated to the existing line. The slope for a complete line is calculated as the average of the slopes of its n-pixel segments.

At the end of the process, the slope tallies are reset to zero and the search continues to the next edge point in the image. This procedure continues until all the edge points in the image have been explored or deleted.

Finally, the local properties of the lines are displayed. The slope of each line is an average of the slopes of the constituent n-pixel long segments. The length of each line is simply determined from its endpoints. These two values (slope and length), along with the endpoints, are displayed for each line in the image that is longer than a user-specified minimum length.

While the starting point of a line segment is identified with as much resolution as possible for the given data, the length of the segment may be off by several pixels. If a line is composed of an integral number of n-pixel long segments, no rounding will occur. Otherwise, if the last segment is less than `min_points`, it will be truncated; and if it is greater than or equal to `min_points`, it will be extended to n pixels. If an application requires greater precision for the length of the segments, the output of Local Lines could be postprocessed to determine the exact length of each segment.

3 Resolution and Complexity

In this section, we compare the resolution and complexity of Local Lines and HTLD. The resolution and complexity of the HTLD is a function of the number of intervals chosen when quantizing the chosen parameters. Let us choose the parameter pair ρ and θ as a basis for comparisons. In this section, we show that the number of possible intervals for ρ and θ in Local Lines is the maximum number that can be justified for pixelized data. Despite this maximal resolution, the complexity of Local Lines for an $M \times N$ image is only $O(M \times N)$, or $O(P)$, where P is the number of pixels in the image. In this section, we first examine the resolution of Local Lines. We then compare the complexity of Local Lines with the complexity of HTLD with similar resolution.

First, consider the number of slopes that can be justified for pixelized data. When scanning an image row by row (from the top of the image to the bottom), as in Local Lines, the possible lines starting at a given pixel (i, j) would extend to an endpoint located below or horizontally to the right of the start point. Any lines that extend above or horizontally to the left of (i, j) would not *start* at (i, j); they would have started at a pixel that was previously examined. As seen in Figure 5, the number of possible slopes for a 10-pixel line starting at (i, j) is 40. We assert that more slopes for pixelized data could not be justified as there are only 40 possible endpoints for the given start point. Similarly, a 20-pixel line has 80 possible endpoints. In general, the maximum number of possible endpoints (and possible number of slopes) is $4n$, where n is the length of the line in pixels.

The resolution for Local Lines is illustrated below with a case where $n = 10$. In this case, Local Lines has 40 possible slopes for a 10-pixel line. When 10-pixel long lines are catenated, we average their respective slopes to derive the slope of the entire line. Thus, for a 20-pixel line, there are 80 possible slopes. For instance, if the first 10-pixel segment has a slope of 20 and the second 10-pixel segment has a slope of 21, the slope for the 20-pixel line would be 20.5. Thus the possible slopes for a 20-pixel line are $(0, 0.5, \ldots, 39.5)$. Furthermore, a 30-pixel line would have possible slopes of $(0, 0.33, \ldots, 39.67)$, or 120 possible slopes. In general, for m 10-pixel segments, the possible slopes are $(0/m, 1/m, \ldots, (40m - 1)/m)$, or $40m$ possible slopes. For n divisible by 10, $4n = 40m$. Therefore, Local Lines offers the maximum number of justifiable slope intervals for pixelized data. We observe that the intervals for θ are not perfectly distributed for our mask. However, it should be noted that even a semi-circular mask would not distribute the intervals uniformly, due to pixelization.

With regard to the second parameter ρ, the number of possible lengths of normals to the origin in Local Lines is the maximum that can be justified as the lines are allowed to be detected anywhere in the image. Therefore, the resolution of Local Lines is the maximum that can be justified for pixel data.

Using a standard HTLD algorithm such as the one in Figure 3, the HTLD yields a complexity of $O(M \times N \times T)$, where T is the number of discretized values of θ. When HTLD encounters an edge point, T calculations are performed. The range for θ is $[0, \pi)$. Thus, to achieve a resolution similar to Local Lines, an $M \times N$ image would require $T = 2(M + N)$, as explained next.

Consider the lines that could be located such that the length of their normals to the origin is the maximum possible for an $M \times N$ image. For example, a vertical line at the right hand edge of the image would have a normal of length N and this normal would intersect the line when $\theta = 0$ at the point (M, N). This is assuming that our coordinate system is based on using the top left corner of the image as origin, with increasing row index towards the bottom and increasing column index towards the right side of the image. It is possible to have a line whose normal intersects it at the point $(M-1, N)$, with a corresponding $\theta > 0$. For $0 \leq \theta < \pi/2$, we see that for each pixel along the right hand side and top row, it is possible to have the intersection of a line and its normal (a few lines in the corner will not be considered as they would be very short). We will get the same number of possibilities with $\pi/2 \leq \theta < \pi$. In order to ensure maximum resolution, we need to have a unique value of θ for each of these lines. Therefore, for maximum resolution, $T = 2(M+N)$.

It follows that $M \times N \times T = 2MN(M+N)$. Hence, the complexity of HTLD for resolution similar to LOCAL LINES is $O(MN(M+N))$. This is substantially more than LOCAL LINES' complexity of $O(MN)$. For instance, for square images, $M = N$ and the complexity of HTLD is $O(N^3)$, while the complexity of LOCAL LINES is $O(N^2)$.

LOCAL LINES, like the HTLD, scans the image to identify each edge pixel. When an edge point is encountered, a constant-time procedure is performed, yielding linear complexity. When HTLD encounters an edge point, a $O(T)$ procedure is performed. Therefore, the complexity of LOCAL LINES is substantially less than that of the HTLD for similar resolution. Furthermore, LOCAL LINES deletes the edge points that compose the found lines. Thus only a fraction of the edge points are operated on. Finally, LOCAL LINES' output does not require any further processing to identify the actual lines in the image.

The space required by LOCAL LINES is dependent on the number of line objects allocated in the line array. If more lines are found in an image than are allocated in the array, an error message is output and the array size can be increased. For applications in which space is highly constrained and very short lines are of little interest, the mask size could be increased and the space complexity would be substantially reduced. For instance, if lines that are shorter than twenty pixels are of little use, a 22×41 mask could be used. While the space complexity of LOCAL LINES is linear in the number of lines detected, the space complexity of HTLD can be problematic, as it is similar to its time complexity.

There are variations of HTLD that reduce time and space requirements. These include parallel processing, dedicated hardware, and software optimizations [5]. According to Leavers [5], software solutions have dominated the research as they are independent of specific hardware requirements. Software solutions include computer friendly algorithms, probabilistic methods, hierarchical approaches, dynamic quantization of accumulators such as the Fast Hough Transform and Adaptive Hough Transform, along with a host of other techniques. These methods offer performance advantages over the standard HTLD for various applications. However, to our knowledge, none offer linear complexity along with maximum resolution. Leavers [5] details benefits and shortcomings of many of these optimizations.

4 Implementation and Results

This program has been implemented in C++. Our code is available by writing to the authors. The following tests were run on Sun Solaris and Linux running on Athlon dual processors. The mask values are read from a text file during the initialization phase of the program.

In Figures 8–10, we show test images in the left column. These images are output by a robotic sensor. In the right column of the images, we graphically display the corresponding results of the Local Lines detector. As seen in the figures, the detector correctly identifies the lines in the image.

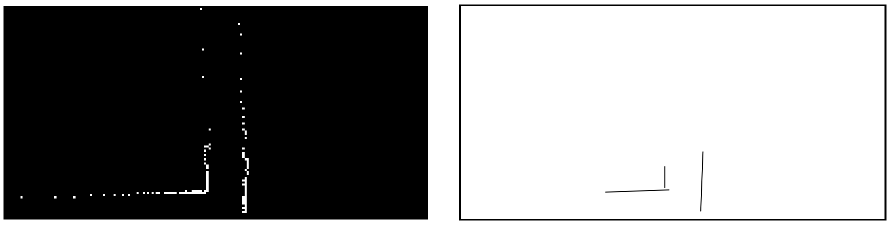

Figure 8: Using Local Lines on test image 1

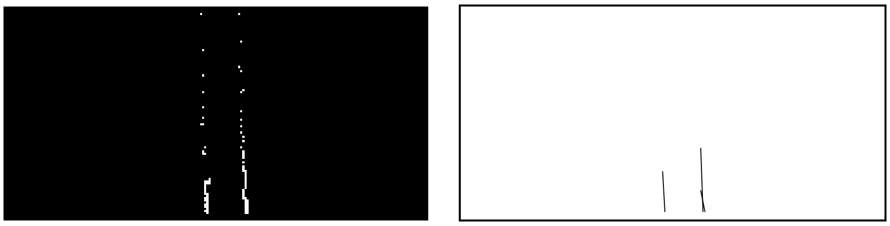

Figure 9: Using Local Lines on test image 2

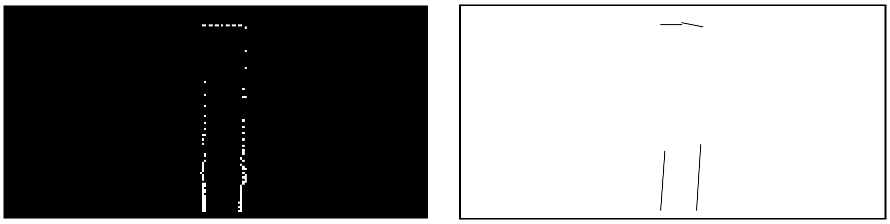

Figure 10: Using Local Lines on test image 4

A set of more complex examples, using photographs, are provided in Figures 11–12. These photographs are first processed using the SUSAN edge detector [7].

Then, the line detector is applied. In the figures, the original photographs are shown in the left column, the preprocessed photographs to show edges are in the middle column, and the output of the line detector is shown in the right column.

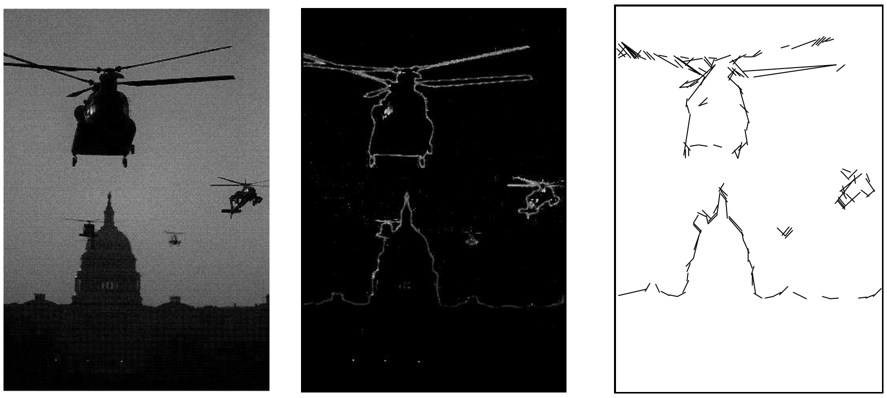

Figure 11: Using Local Lines on Smithsonian test image 1

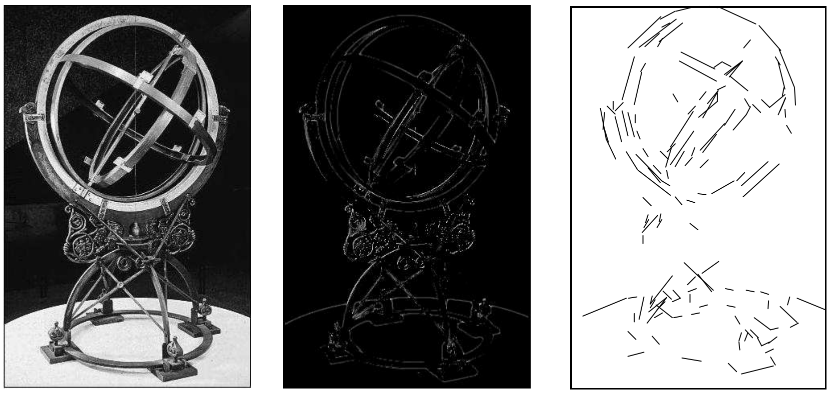

Figure 12: Using Local Lines on Smithsonian test image 2

5 Conclusions and Future Work

The line detection algorithms based on HTLD are typically characterized by high computational costs. Hemdal has put forward an algorithm based on measuring the periodicity of lines in a single dimension by unwrapping the binary image and applying Discrete Fourier Transform but the results are applicable in a limited set of problems [2]. Lagunovsky and Ablameyko have used cluster analysis methods to extract straight lines but the method is sensitive to straightness of lines [4].

We have received encouraging results from LOCAL LINES during our various experiments. We will like to note that both HTLD and LOCAL LINES are robust. However, HTLD's use of global properties results in greater complexity than LOCAL LINES. Moreover, HTLD outputs global properties of lines that require post-processing adding an extra step to the computation. In contrast, LOCAL LINES offers maximum justifiable resolution for pixelized data, while operating in linear time. This efficiency stems from the use of local properties to detect lines in images.

We have identified a few improvements to be made to LOCAL LINES. A simple extension would prevent the identification of gently curved lines as straight lines. We can achieve that by making a comparison just before catenating a new segment to an existing line. If the difference between the slope of the new segment and the original, first n-pixel line segment of the line are greater than a threshold, we would not catenate the new segment.

We are also looking into possible improvement by fine-tuning the mask. In our experiments, we used a 12×21 rectangular mask. We would like to experiment with other sizes and/or shapes of masks. Furthermore, we can adjust the mask values. The current values were rather arbitrarily chosen. We can also employ various constraints on mask values to improve performance. For instance, the sum of weights for each slope could be forced to equal the same value for all the slopes. Another constraint might be to enforce a minimum tally for a specified deterioration of a "perfect" line for each slope.

Finally, we note that for some slopes, there is no set of pixels that constitute a "straight" line. To account for these variations, the values of configurable parameters could be varied for specified slopes, accounting for differences manifested by the representation of sloping lines by pixel data.

Acknowledgement

We thank Andy Martignoli for providing the robotic sensor data that we used in our tests. Bill Smart, Chengjun Liu, and Shawna Climer contributed a number of ideas and some software. The photographs used in Figures 11 and 12 are courtesy of Smithsonian Institute. We will also like to thank the anonymous referees for their comments that helped to improve the paper.

References

[1] P. Ballester. Applications fo the Hough transform. In R. H. D.R. Crabtree and J. Barnes, editors, *Proceedings of the Astronomical Data Analysis Software and Systems Conference III*, 1994.

[2] J. Hemdal. One-dimensional digital processing of images for straight-line detection. *Pattern Recognition*, 31(11):1687–1690, 1998.

[3] R. Jain, R. Kasturi, and B. G. Schunck. *Machine Vision.* McGraw Hill, New York, NY, 1995.

[4] D. Lagunovsky and S. Ablameyko. Straight-line-based primitive extraction in gray-scale object recognition. *Pattern Recognition Letters*, 20:1005–1014, 1999.

[5] V. Leavers. *Shape Detection in Computer Vision Using the Hough Transform.* Springer-Verlag, London, England, 1992.

[6] R. Shpilman and V. Brailvosky. Fast and robust techniques for detecting straight line segments using local models. *Pattern Recognition Letters*, 20:865–877, 1999.

[7] S. Smith and J. Brady. SUSAN - a new approach to low level image processing. *Int. Journal of Computer Vision*, 23(1):45–78, May 1997.

[8] E. Trucco and A. Verri. *Introductory Techniques for 3-D Computer Vision.* Prentice-Hall, Upper Saddle River, NJ, 1998.