
This version is available at https://strathprints.strath.ac.uk/63250/

Strathprints is designed to allow users to access the research output of the University of Strathclyde. Unless otherwise explicitly stated on the manuscript, Copyright © and Moral Rights for the papers on this site are retained by the individual authors and/or other copyright owners. Please check the manuscript for details of any other licences that may have been applied. You may not engage in further distribution of the material for any profitmaking activities or any commercial gain. You may freely distribute both the url (https://strathprints.strath.ac.uk/) and the content of this paper for research or private study, educational, or not-for-profit purposes without prior permission or charge.

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

I. INTRODUCTION

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

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

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

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

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

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

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

II. THE LINE HOUGH TRANSFORM

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

\[ \rho = x\cos(\theta) + y\sin(\theta) \quad (0^\circ \leq \theta^\circ < 180^\circ). \]  

(1)

The HPS represents an accumulator array \(A(\theta, \rho)\), that is pre-allocated in memory. The HPS is quantised with \(N_\theta\) and \(N_\rho\) levels, chosen according to the applicable accuracy requirements and memory constraints. The edge pixels within the image are processed using (1) over a discrete range of \(\theta\) values \([0 : \delta_\theta : \theta_{\text{max}}]\), where \(\delta_\theta\) is the discretisation step and \(\theta_{\text{max}}\) is the maximum value of \(\theta\). The resulting parameters form an address that applies a vote to the corresponding location in the HPS.
The result of voting is to create a sinusoid in the HPS for each edge pixel. Figure 1 demonstrates this relationship using three edge pixels on a line, $E_0(x_0, y_0)$, $E_1(x_1, y_1)$, $E_2(x_2, y_2)$, and their corresponding Hough Parameters as calculated using (1). The sinusoids produced by edge pixels that exist on line $i$ within the image intersect at a common point $(\theta_i, \rho_i)$.

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

**A. HPS Memory Requirements**

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

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

(2)

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

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

$$A_{\text{bits}} = N_{\rho}N_{\theta}\left[\log_2(2\rho_{\text{max}})\right].$$

(3)

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

**III. THE PROPOSED METHOD**

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

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

![Fig. 1: LHT mapping between edge image (left) and HPS (right).](image1.png)

![Fig. 2: Overview of the AR-LHT algorithm.](image2.png)

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

**A. Calculating the Edge Gradient Orientation**

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

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

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

(4)

The edge gradient orientation can then be computed using

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

(5)

The value of $\theta_G$ is brought into the limits of (1) by adding $180^\circ$ if $\theta_G < 0$. It can then be used in (1) as $\theta$ to calculate $\rho$.

**B. Optimising the HPS**

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

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

$$A_{\text{bits}}^{\text{opt}} = \frac{N_{\rho}N_{\theta}\left[\log_2(2R\rho_{\text{max}})\right]}{R}.$$  

(6)

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

![Fig. 3: Graph of the HPS.](image3.png)
Votes can only be applied to the reduced HPS if they are in the range \([0,(\theta_{\text{max}} + \delta_{\theta})/R - \delta_{\theta}]\). To ensure each vote is applied to the HPS, inapplicable votes are adjusted into the above range. This adjusted orientation is named \(\theta_A\) and, while voting, is applied to the HPS with the same value of \(\rho\).

### C. Calculating the Adjusted Orientation \(\theta_A\)

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

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

### D. Separating Peaks using the RBM

The RBM is an array of size \(N_{\theta} \times N_{\rho} \times 1\), that is set to zero before processing begins. The RBM is required to keep a record of \(\theta_G\), and \(\rho\), for all of the edge pixels that have been processed using (1). It achieves this by using \((\theta_G, \rho)\) as an index, and changing the value of that location to a binary ‘1’.

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

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

### IV. IMPLEMENTATION

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

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

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

The Hough parameters, \(\rho\) and \(\theta_G\), are used to generate an address to access the RBM and update it accordingly. Similarly the HPS uses \(\rho\) and the adjusted angle \(\theta_A\) to generate an address to accumulate votes.

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

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

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

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

### V. PERFORMANCE AND RESOURCE ANALYSIS

The AR-LHT was synthesesed and implemented using Viva Design Suite for various image resolutions, and shown to achieve a clock frequency of 145MHz on a Xilinx xc7z010-1 device. Table I provides the memory consumption, in bits, for each tested resolution and compares them to the memory requirements of the Standard LHT, calculated using (6).
The standard LHT requires 2,871,000 bits of memory for an image of 1024x1024 pixels. The xc7z010-1 device would be unsuitable for implementation as it only contains 2,211,840 bits of dedicated FPGA memory. However, it can apply the AR-LHT to the same image resolution due to its memory-efficient HPS. The AR-LHT requires 1,474,560 bits, saving 48.64% of memory as compared to the Standard LHT.

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

Table II provides implementation results of this work and the results from other published FPGA implementations of the LHT, three of which targeted Altera devices. As Xilinx and Altera devices are not directly comparable for total resource requirements in comparison to similar work. An image of 512x512 pixels, at the expense of high memory consumption and execution time.

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

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

Guan et al. [8] created a system which applied the LHT to a video stream. They reduced the total memory requirements of the HPS, by setting the discretisation step of $\theta$ to $2^\circ$ ($\delta_\theta = 2^\circ$). This has the effect of reducing the accuracy of line detection, unlike our work, which still achieves $\delta_\theta = 1^\circ$. Even for unequal $\delta_\theta$, the AR-LHT provides a memory saving of 16%. Lu et al. [10] developed a shift-and-add LHT architecture with $\delta_\theta$ set to 0.8952°. This achieved better accuracy of extracted lines whilst increasing the memory requirements of the HPS. If this approach was combined with the AR-LHT (a possible avenue of future work), the HPS could become more memory-efficient while also improving accuracy.

### VI. CONCLUSION

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

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

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

<table>
<thead>
<tr>
<th>Resolution (pixels)</th>
<th>Execution Time (ms)</th>
<th>Memory (Bits)</th>
<th>% Saved</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024x1024</td>
<td>9.03ms</td>
<td>1,474,560</td>
<td>48.64</td>
</tr>
<tr>
<td>512x512</td>
<td>2.71ms</td>
<td>442,368</td>
<td>66.15</td>
</tr>
<tr>
<td>1024x768</td>
<td>7.01ms</td>
<td>1,345,536</td>
<td>67.01</td>
</tr>
<tr>
<td>48.64</td>
<td></td>
<td>421 (Registers)</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Device</th>
<th>Resolution (pixels)</th>
<th>Frequency</th>
<th>Execution Time (ms)</th>
<th>Logic Blocks</th>
<th>DSPs</th>
<th>Memory (Bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed Method</td>
<td>Xilinx XC7Z010-1</td>
<td>1024x1024</td>
<td>154MHz</td>
<td>9.03ms (110fps)</td>
<td>2202 (Slices)</td>
<td>0</td>
<td>1,474,560</td>
</tr>
<tr>
<td>Zhou et al. [9]</td>
<td>Xilinx XC6VLX240T-1</td>
<td>512x512</td>
<td>200MHz</td>
<td>1.00ms (936fps)</td>
<td>14493 (Slices)</td>
<td>178</td>
<td>3,317,760</td>
</tr>
<tr>
<td>Chen et al. [14]</td>
<td>Altera EP2S180F1508C3</td>
<td>512x512</td>
<td>200MHz</td>
<td>2.07-3.61ms (277-483fps)</td>
<td>855 (ALUTs)+421 (Registers)</td>
<td>0</td>
<td>3,270,032 (off-chip)+223,360 (on-chip)</td>
</tr>
<tr>
<td>Guan et al. [8]</td>
<td>Altera EP4SGX230KF40C2</td>
<td>1024x768</td>
<td>200MHz</td>
<td>5.4ms (185fps)</td>
<td>1196 (Registers)</td>
<td>32</td>
<td>1,585,202 (on-chip)</td>
</tr>
<tr>
<td>Lu et al. [10]</td>
<td>Altera EP4CE115F29</td>
<td>1024x768</td>
<td>200MHz</td>
<td>15.59ms (64fps)</td>
<td>15,704 (LEs)</td>
<td>8</td>
<td>3,052,344</td>
</tr>
</tbody>
</table>
REFERENCES


