A PYNQ Evaluation Platform for FPGA Architectures of the Line Hough Transform

David Northcote*, Louise H. Crockett, Paul Murray and Robert W. Stewart
Department of Electronic and Electrical Engineering, University of Strathclyde, Glasgow, Scotland, UK
*david.northcote@strath.ac.uk

***Abstract*—The Line Hough Transform (LHT) is an effective line detection algorithm for digital images. To meet real-time requirements, Field Programmable Gate Arrays (FPGAs) are often chosen to accelerate the LHT. However, many LHT architectures select different design parameters and discretisation steps for the Hough Parameter Space (HPS). The work presented in this paper describes a novel evaluation platform for FPGA architectures of the LHT. Our system is named the Hough Evaluation Platform (HEP) and can be used to visualise and inspect the HPS produced from LHT architectures that use different design parameters. Architectures are compared by evaluating the HPS using a unique measurement named the Peak to Mean Vote Ratio (PMVR). Our system employs the PYNQ framework on a Xilinx Zynq MPSoC device for the visualisation of the HPS and also determines the processing time of a given LHT architecture. The HEP has been implemented on a XCZU7EV-2e device and can operate up to a target frequency of 250 MHz.**

I. Introduction

Line extraction is essential for a wide range of vision applications such as Advanced Driver Assistance Systems (ADAS) [1] and Unmanned Aerial Vehicle (UAV) surveillance [2]. The Line Hough Transform (LHT) is commonly selected to extract lines from digital images as it is robust to noise and can detect lines that are partially occluded [3], [4]. The LHT performs line extraction by mapping edge pixels to a discrete array known as the Hough Parameter Space (HPS). The parameters of edge pixels are used to apply a 'vote' to the HPS, with votes accumulating to form peaks. The coordinates of peaks that are extracted from the HPS correspond to the parameters of lines in the original image.

The LHT is a computational algorithm that typically demands dedicated processing resources for real-time applications. Optimisations have been presented in past work to reduce the complexity of the LHT in software implementations [5], [6], [7], [8]. However, to meet demanding time constraints, the LHT is commonly implemented on a Field Programmable Gate Array (FPGA). In [9] a pipelined shift-and-add architecture is presented. The authors in [10] implemented a low-resource application specific architecture for ADAS. In [11] a low-latency architecture is described that can apply the LHT to an image in 1.07ms. In [12], a memory-efficient architecture of the LHT is presented.

Many of the previously reported LHT architectures use varying design parameters and image resolutions, which make them difficult to compare. Furthermore, verifying that an architecture of the LHT has operated correctly is also challenging, as the HPS needs to be retrieved and inspected.

In this paper, we present a novel evaluation platform for the rapid inspection of LHT architectures. We name this system the Hough Evaluation Platform (HEP). Many of the previously reported FPGA implementations of the LHT use simulations to test architecture designs. Instead, the HEP allows users to visualise and inspect the HPS produced as a result of applying an image to an LHT architecture on the physical target device. This improves testing of the hardware architecture beyond software simulations, and provides a very powerful technique for architecture validation. The HEP is also capable of accurately calculating the time taken for an LHT architecture to process an image.

We also describe how to evaluate and compare LHT architectures that have different discretisation steps for the HPS. We name this measurement the Peak to Mean Vote Ratio (PMVR), which is obtained by calculating the ratio of the largest peak to the mean votes of all the non-zero locations in the HPS. It is useful for quantifying the ability to reliably extract and separate peaks in the HPS from other neighbouring locations. The PMVR is calculated by applying a test image to the LHT architecture being evaluated, and can be used to compare against other LHT architectures that have been designed using different parameters and HPS discretisation steps. The PMVR measurements are included in the HEP, providing effective feedback of the LHT architecture under evaluation.

The HEP is implemented using the Xilinx Zynq MPSoC device and employs the PYNQ (Python Productivity for Zynq) [13] framework to leverage its signal tracing, visualisation, and control capabilities. The PYNQ software stack operates in the Zynq MPSoC's Processing System (PS), where users can communicate with an interactive Jupyter Lab [14] session via network connection using a standard web browser. The LHT architecture and performance analysis tools operate in the Zynq MPSoC's Programmable Logic (PL), which contains FPGA logic fabric for implementing hardware accelerators.

The primary aim of this work is to create a common evaluation tool for custom FPGA architectures of the LHT. For this reason, the design files for the HEP are available to download online in [15]. This framework will allow the research community to likewise evaluate their own LHT architectures using PYNQ.

The remainder of this paper is organised as follows: Section II details the LHT algorithm, Section III presents the architecture of our novel HEP, Section IV evaluates and analyses the HEP, and Section V concludes this paper.

II. The Line Hough Transform

The LHT operates on a binary edge image that is obtained by applying an edge detection operator, such as Sobel [16], to a greyscale image. Lines are detected by exploiting the relationship between collinear sets of edge pixels and a single point in an associated HPS. For each edge pixel, a corresponding set of parameters is derived. These are an edge pixel's magnitude of displacement ρ and orientation of displacement θ from a predefined image origin, as given in (1).

$$\rho(\theta) = xcos(\theta) + ysin(\theta) \tag{1}$$

The Hough parameters (ρ, θ) are used to vote in the HPS, which is a discrete two-dimensional array $A(\rho, \theta)$ that is set to zero before voting begins. The HPS is quantised over N_ρ and N_θ levels, where ρ and θ each have their own discretisation steps, δ_ρ and δ_θ respectively.

Voting is carried out by applying (1) over a discrete range of θ values $[0 : \delta_\theta : \theta_{max}]$, where θ_{max} is the maximum value of θ. The resulting Hough parameters (ρ, θ) form an address that is used to access a location in the HPS. A vote is applied to that location by incrementing its stored value by one. This has the effect of creating sinusoids in the HPS, which accumulate to form local peaks.

Figure 1 illustrates the relationship between edge pixels in the spatial image domain and their corresponding Hough parameters in the HPS. As shown, a peak has formed at (ρ_i, θ_i) in the HPS, which corresponds to the parameters of a line i in the original edge image. After voting is complete, the HPS is searched for parameters that have accumulated a significant number of votes. Line reconstruction can be performed in the spatial image domain by applying the Hough parameters to the inverse function of (1).

Due to the computational complexity of the LHT, many reported FPGA architectures modify the quantisation interval, δ_θ, and deliberately modify the wordlength or size of the HPS to affect the memory consumption [9], [17]. For example, the architecture presented in [18] sets $\delta_\theta = 2°$, while the architecture described in [9] sets $\delta_\theta = 0.8952°$. A method is required to compare these architecture that have different design parameters for the HPS. Our novel HEP is capable of analysing the results of the LHT architectures and visualising the HPSs.

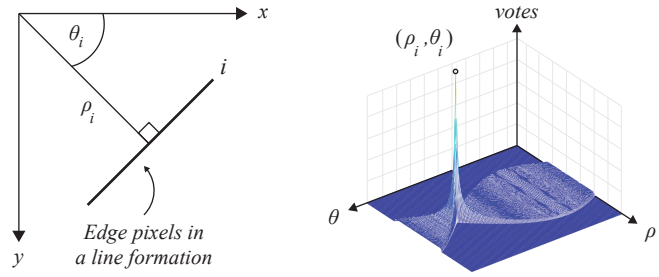

Fig. 1. Edge pixels in the spatial image domain (left) and their corresponding Hough parameters in the HPS (right).

III. Overview of the HEP

In this section, we describe our novel platform, the HEP, which evaluates FPGA architectures of the LHT. An overview of the proposed HEP is given in Figure 2. The HEP is implemented on the Xilinx Zynq MPSoC device, where the PL is used to implement the Hough Inspection Unit (HIU) and the PS is host to the PYNQ software framework. The primary goal of the HEP is to evaluate the resulting HPS of the LHT architecture under test. Therefore, it is not necessary to reconstruct lines using the Zynq's PL.

The HIU contains two Advanced eXtensible Interface (AXI) Direct Memory Access (DMA) cores. The input DMA transfers a candidate image from off-chip memory to the FPGA so that it can be processed by the Design Under Test (DUT). The DUT contains a custom LHT architecture that is under evaluation. The output DMA transfers the resulting HPS from the FPGA into external memory. While the candidate image

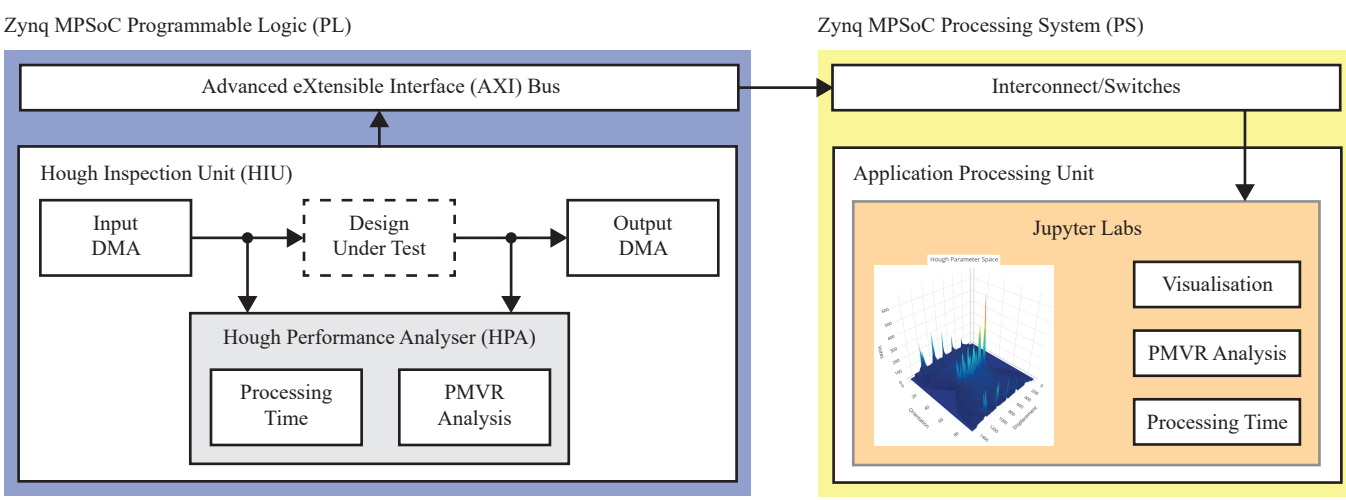

Fig. 2. System architecture of the proposed HEP. The Design Under Test block contains the custom LHT architecture to be evaluated by the HIU.

is passed through the LHT architecture, a Hough Performance Analyser (HPA) unit calculates the processing time of the LHT architecture and the PMVR of the resulting HPS.

Once an image has been processed by the LHT architecture, the corresponding HPS is analysed in Jupyter Lab, which operates in the PS of the Zynq MPSoC. As Jupyter Lab uses Python, the HPS can be visualised using the interactive plotting library known as Plotly [19]. It can also be inspected for peaks and manipulated by the user as required. The performance measurements calculated in the HPA are also presented to the user. The HPS and performance measurements can be saved in memory and compared to a different LHT architecture as required.

The remainder of this section describes how to inspect and visualise the HPS in Jupyter Lab. The method used to calculate the processing time of the LHT architecture is also described. Finally, we present the PMVR, which can be used to compare LHT architectures that have been designed using different parameters and discretisation steps for the HPS.

A. HPS Visualisation and Inspection

Upon passing an image to the LHT architecture, the resulting HPS is generated and analysed in Jupyter Lab. The HPS is displayed using the interactive plotting library, Plotly. In particular, the application programming interface (API) used to visualise the HPS is plotly.surface. An example plot of the HPS is shown in Figure 3, where the user can freely zoom, pan, and orbit the plot as required.

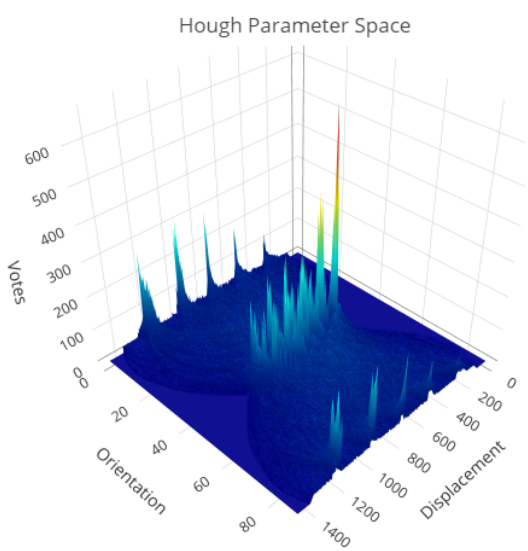

Fig. 3. A plot of the HPS using the Python Plotly library.

It is important to note that the HPS displayed using Plotly in Figure 3 was generated from the LHT architecture operating in the PL of the Zynq MPSoC. This means that the architecture is operating in real-time using fixed-point arithmetic. Testing the LHT architecture in this way is useful to ensure correct operation on native hardware and is a far more effective technique for design validation than software simulations.

The general flow to plot the HPS of an image begins by transferring a candidate image to the LHT architecture using the input DMA. When processing is complete the resulting HPS is transferred to off-chip memory via the output DMA. At this stage the user can optionally pre-process the HPS using the Jupyter Lab environment to manipulate it as required. The HPS is then plotted using Plotly's plotly.surface API.

In addition to plotting the HPS produced by an LHT architecture, it is possible to create a software model to compare against the hardware result, using Jupyter Lab and Python. Testing and comparing software and hardware implementations increases the effectiveness of the system to validate FPGA architecture designs of the LHT.

B. Processing Time Calculation

The input DMA transfers image data using the AXI-Stream interface, which requires 4 signals to successfully transfer data in our proposed HEP. These signals are as follows: *tdata* transfers image data to the LHT architecture, *tvalid* indicates the presence of valid data, *tlast* indicates the last data beat in the transfer, and *tready* informs the input DMA that the LHT architecture is ready to receive data. Similar signalling is performed when transferring HPS data to the output DMA.

The processing time of the architecture is calculated using several hardware counters that maintain the time spent operating on an image. The counters begin when a rising edge is detected on the *tvalid* signal on the input DMA. The counters stop when the *tlast* signal sent to the output DMA is high. This technique ensures an accurate representation of the architecture's processing time, and can be used to find the processing rate that the architecture can achieve, in frames per second (fps).

C. PMVR Analysis

The PMVR is our novel technique of quantifying the ability to reliably extract and separate peaks in the HPS. Initially, the mean votes in the non-zero locations of the HPS, N_f, is calculated. This is performed by dividing the total number of votes applied to $A(\rho, \theta)$ by the total number of non-zero locations present in the HPS. The PMVR, denoted as R_f, can then be found using (2).

$$R_f = \frac{max(A(\rho, \theta))}{N_f} \tag{2}$$

In the current implementation of the HEP, after applying the LHT to an image, the PMVR is presented to the user as a single floating-point value. In order to demonstrate the effectiveness of the PMVR as a quantitative measurement, which describes the ability to reliably extract and separate peaks in the HPS, two LHT examples were created using MATLAB. These are shown in Figure 4, where the left plot presents the HPS after applying the LHT to an image using $\delta_\theta = 1°$, while the right plot presents the HPS for the same image when $\delta_\theta = 2°$. Each plot has been rotated to only show the Votes and Orientation axes.

Notably, the plot on the left of Figure 4 has its largest peak in the centre of the HPS, while the peak in the right plot has decreased in size. This has occurred due to the different sizes of δ_θ used to create the respective HPSs, and causes votes to spread away from their optimal locations. This may cause inaccuracies when reconstructing lines in the original edge image.

The mean votes in the non-zero locations of the HPS, N_f, closely approximates the mean number of votes from small sets of collinear edge pixels. These sets of edge pixels are the cause of noise in the HPS and can affect the extraction of the greatest peaks if the noise is large enough.

As shown in Figure 4, the PMVR is larger when $\delta_\theta = 1°$. This is because there are more locations in the HPS for votes to spread into and accumulate peaks. Locations that have a low number of votes either indicate a small set of collinear edge pixels in the original image, or inform us that the optimal location for an edge pixel was not available and instead, it accumulated in the closest available location of the HPS. When $\delta_\theta = 2°$, the largest peak decreased in size, causing a lower value of PMVR.

IV. Evaluation of the HEP

The HEP was synthesised and implemented using the Xilinx Vivado Design Suite and achieved a target clock frequency of 250 MHz on a Xilinx XCZU7EV-2e device. To effectively analyse the FPGA resources consumed by the HEP alone, it was implemented separately, i.e. an LHT architecture was excluded from the design. The FPGA resources used by the HEP were reported as shown in Table I. Notably, the HEP consumes very few FPGA resources and only uses 17 Block RAM tiles for both of the AXI DMA blocks. The remaining FPGA resources are available for a custom LHT architecture to be implemented.

TABLE I
XCZU7EV-2e FPGA resource requirements for the HEP.

Resource	Available	Used	Percentage (%)
LUTs	230400	4155	1.80
LUTRAM	101760	304	0.30
Flip-Flops	460800	5076	1.10
Block RAM	312	17	5.45
DSP48E2	1728	0	0.00

To further test the HEP, a custom architecture that is capable of applying the LHT to an image of 1280x720 pixels was designed. The HPS was designed to use $\delta_\rho = 1$, $\delta_\theta = 1°$, and the memory was not constrained in any way that would cause votes to be applied incorrectly i.e. the HPS was not deliberately reduced in size.

To simplify the integration of the LHT architecture with the HEP, a Mathworks' HDL Coder reference design that automatically generates a Vivado Design Suite project was created. This allows the user to design their reference design using a Simulink template and follow the HDL Coder Workflow Advisor to rapidly generate and test their LHT architecture using the HEP. When a bitstream is generated, it is transferred to the PYNQ framework for evaluation.

The FPGA resources consumed by LHT architecture and HEP, when targeting the XCZU7EV-2e device, were reported as shown in Table II.

TABLE II
XCZU7EV-2e FPGA resource requirements for the HEP and LHT architecture.

Resource	Available	Used	Percentage (%)
LUTs	230400	17525	7.61
LUTRAM	101760	286	0.28
Flip-Flops	460800	12065	2.62
Block RAM	312	197	63.14
DSP48E2	1728	89	5.15

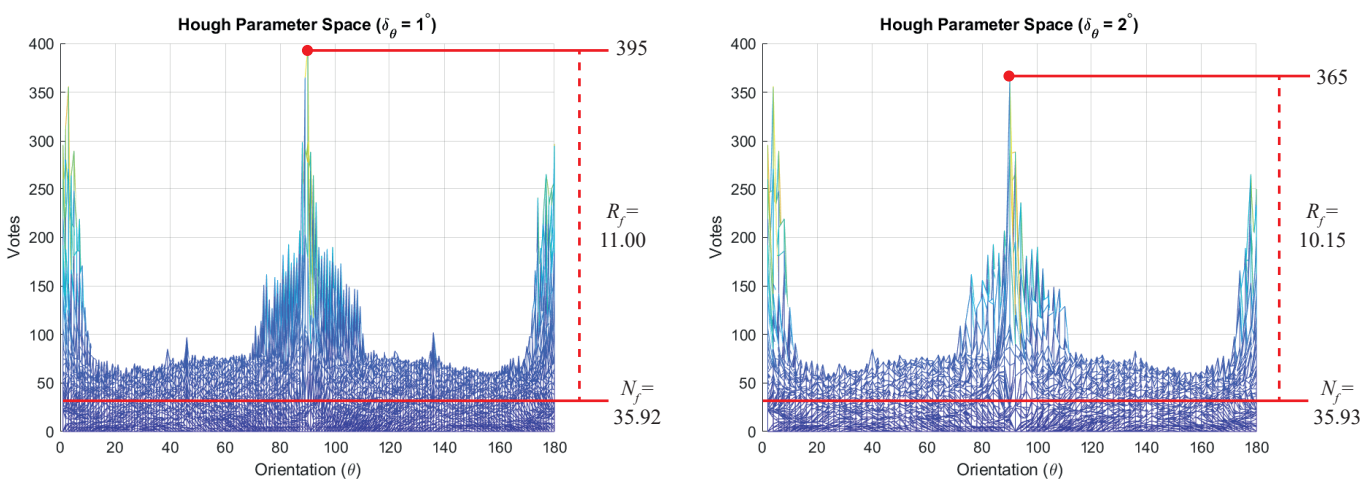

Fig. 4. Plots of the HPS after applying the LHT to an image using $\delta_\theta = 1°$ (left) and $\delta_\theta = 2°$ (right). Each plot has been rotated to only show the Votes and Orientation axes. The PMVR is higher when $\delta_\theta = 1°$.

This architecture was able to achieve a target clock frequency of 150 MHz. Using the HEP, the custom architecture applied the LHT to an input image resulting in a corresponding HPS. The HPS was correctly displayed and inspected as previously shown in Figure 3. The processing time reported by the HEP was 7.91 ms, which corresponds to 126.45 fps.

To evaluate the PMVR, the LHT was applied to three different images [15] using various values of δ_θ. The results presented in Figure 5, demonstrate that the PMVR is strongest when δ_θ is less than 1. Generally, the PMVR decreases as δ_θ increases. Although a correlation between PMVR and δ_θ has been identified, we recognise that further tests are required to fully explore the impact of PMVR and peak extraction. Further work is required to compare the PMVR and other design parameters such as the HPS wordlength, HPS size, and the value of δ_ρ. In the current implementation of the HEP, the PMVR provides an interesting quantitative measurement to compare the HPSs produced from different LHT architectures.

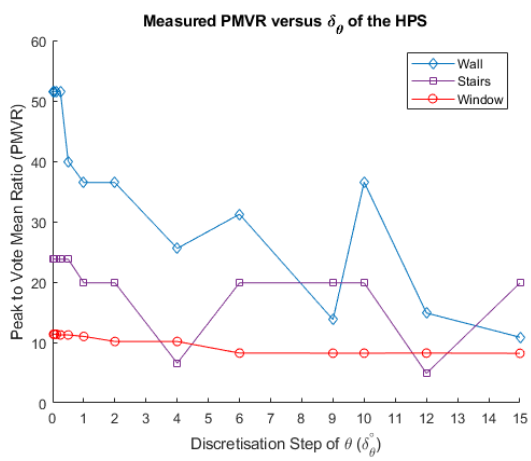

Fig. 5. A plot of the measured PMVR across several values of δ_θ.

V. Conclusions

This paper has presented the HEP, a rapid evaluation platform for FPGA architectures of the LHT based on PYNQ. An overview of the HEP was provided and it was shown to be capable of plotting the HPS produced from applying an image to a custom LHT architecture. The HEP was also shown to calculate the time taken to process an image.

The PMVR measurement unit implemented in the HEP proved effective when determining the ratio of the largest peak to the mean votes of all non-zero elements in the HPS. Further work should be undertaken to analyse the effectiveness of the PMVR for the evaluation of LHT architectures.

The HEP architecture was developed using Mathworks' HDL Coder and the Xilinx Vivado Design Suite. It was implemented onto a XCZU7EV-2e Zynq MPSoC device and achieved a target clock frequency of 250 MHz. An LHT architecture was also developed and successfully evaluated using the HEP.

Design files for the HEP have been made available in [15], which will allow others in the research community to also evaluate their LHT architectures using the PYNQ framework described in this paper.

Acknowledgement

The authors would like to acknowledge funding, hardware, and software support from Xilinx, Inc. The authors acknowledge funding for David Northcote under EPSRC grant no. EP/M508159/1.

References

[1] T. F. Bente, S. Szeghalmy, and A. Fazekas, "Detection of lanes and traffic signs painted on road using on-board camera," in *2018 IEEE International Conference on Future IoT Technologies (Future IoT)*, Jan 2018, pp. 1–7.

[2] A. I. Purica, B. Pesquet-Popescu, and F. Dufaux, "A railroad detection algorithm for infrastructure surveillance using enduring airborne systems," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2017, pp. 2187–2191.

[3] P. V. C. Hough, "Machine analysis of bubble chamber pictures," *2nd International Conference on High-Energy Accelerators and Instrumentation*, vol. 73, pp. 554–558, 1959.

[4] R. O. Duda and P. E. Hart, "Use of the Hough transform to detect lines and cures in pictures," *Communications of the Association Computing Machinery*, vol. 15, no. 1, pp. 11–15, 1972.

[5] L. Xu, E. Oja, and P. Kultanen, "A new curve detection method: Randomized Hough Transform (RHT)," *Pattern Recognition Letters*, vol. 11, no. 5, pp. 331–338, 1990.

[6] N. Kiryati and Y. Eldar, "A Probabilistic Hough Transform," *Pattern Recognition*, vol. 24, no. 4, pp. 303–316, 1991.

[7] J. Matas, C. Galambos, and J. Kittler, "Robust Detection of Lines Using the Progressive Probabilistic Hough Transform," *Computer Vision and Image Understanding*, vol. 78, no. 1, pp. 119–137, 2000.

[8] L. A. F. Fernandes and M. M. Oliveira, "Real-time line detection through an improved Hough transform voting scheme," *Pattern Recognition*, vol. 41, no. 1, pp. 299–314, 2008.

[9] X. Lu, L. Song, S. Shen, K. He, S. Yu, and N. Ling, "Parallel Hough Transform-based straight line detection and its FPGA implementation in embedded vision." *Sensors (Basel, Switzerland)*, vol. 13, no. 7, pp. 9223–9247, 2013.

[10] I. E. Hajjouji, S. Mars, Z. Asrih, and A. E. Mourabit, "A novel fpga implementation of hough transform for straight lane detection," *Engineering Science and Technology, an International Journal*, vol. 23, no. 2, pp. 274 – 280, 2020.

[11] X. Zhou, N. Tomagou, Y. Ito, and K. Nakano, "Efficient hough transform on the FPGA using DSP slices and block RAMs," *Proceedings - IEEE 27th International Parallel and Distributed Processing Symposium Workshops and PhD Forum, IPDPSW 2013*, pp. 771–778, 2013.

[12] D. Northcote, L. H. Crockett, and P. Murray, "FPGA Implementation of a Memory-Efficient Hough Parameter Space for the Detection of Lines," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5.

[13] Xilinx, Inc. PYNQ: Homepage. [Online]. Available: https://pynq.io

[14] Project Jupyter. JupyterLab Documentation. [Online]. Available: https://jupyterlab.readthedocs.io/

[15] D. Northcote (Creator), L. H. Crockett (Contributor), P. Murray (Contributor), and R. W. Stewart (Contributor). (2020, May) The Line Hough Transform Evaluation Platform. [Online]. Available: https://github.com/dnorthcote/lht_framework

[16] I. Sobel, "An isotropic 3 by 3 image gradient operator," *Machine Vision for three-demensional Sciences*, vol. 1, no. 1, pp. 23–34, 1990.

[17] D. G. Bailey, "Streamed Hough Transform and Line Reconstruction on FPGA," *International Conference Image and Vision Computing New Zealand*, vol. 2017-Decem, pp. 1–6, 2018.

[18] J. Guan, F. An, X. Zhang, L. Chen, and H. J. Mattausch, "Real-time straight-line detection for XGA-size videos by hough transform with parallelized voting procedures," *Sensors (Switzerland)*, vol. 17, no. 2, 2017.

[19] Plotly. Plotly python open source graphing library. [Online]. Available: https://plot.ly/python/