Lane Detection using OpenCV on RISC-V based PolarFire SoC Icicle kit

Sanket Patadiya, Mohit Sapkal, Krishil Gandhi,
Vivek Chauhan, Umair Jarullah, Sohan Patel

January 6, 2023

Abstract

The *lane detection algorithms* in "Advanced Driver Assistance Systems" (ADAS), are one of the algorithms that use image processing techniques. Unfortunately, the image processing algorithms are slow on processors because of the processing intensity of image processing techniques. High-priced hardware, such as the "Graphics Processing Unit" (GPU), is used to accelerate this slow operation. However, the usage of GPU in projects increases project costs, so the end user is offered a product at a high price. This situation reduces the usability of image processing techniques and applications in daily life.

In this project, the PolarFire SoC Icicle kit of Microchip Inc. is used as a "System on Chip" (SoC) to solve this problem. A lane detection algorithm has been developed as an image processing project, and it was implemented in the FPGA after this developed algorithm was broken down into software and hardware.

1 Introduction

Road safety has been a significant concern in the automotive domain. In 2021 in India, over 4,12,432 road accidents have been reported by the Police Department of States and Union Territories, claiming 1,53,972 lives and causing injuries to 3,84,448 persons. Most road accidents occur due to human error, such as overspeeding, drunken driving, wrong lane driving, jumping at a red light, or a driver's low reaction time[8].

Advance Driver Assistance System (ADAS) has become indispensable for top car manufacturers, so much so that each develops its own ADAS system. Nowadays, ADAS is a primary research topic in the automotive research field. Projects such as Tesla autopilot or Google car Waymo are sample achievements in this field and a testimony that it is still an unfinished task. ADAS's current industrial implementation is restricted to high-end vehicles in contrast to the primary goal of this research, which is to implement ADAS in mid-range vehicles[5]. The three major sensors self-driving cars use together as the human eyes and brain. These sensors are cameras, radar, and lidar. Together, they give the car a clear view of its environment. They help the car identify the location, speed, and 3D shapes of objects close to it. Self-driving cars are now being built with inertial measurement units that monitor and control acceleration and location[1].

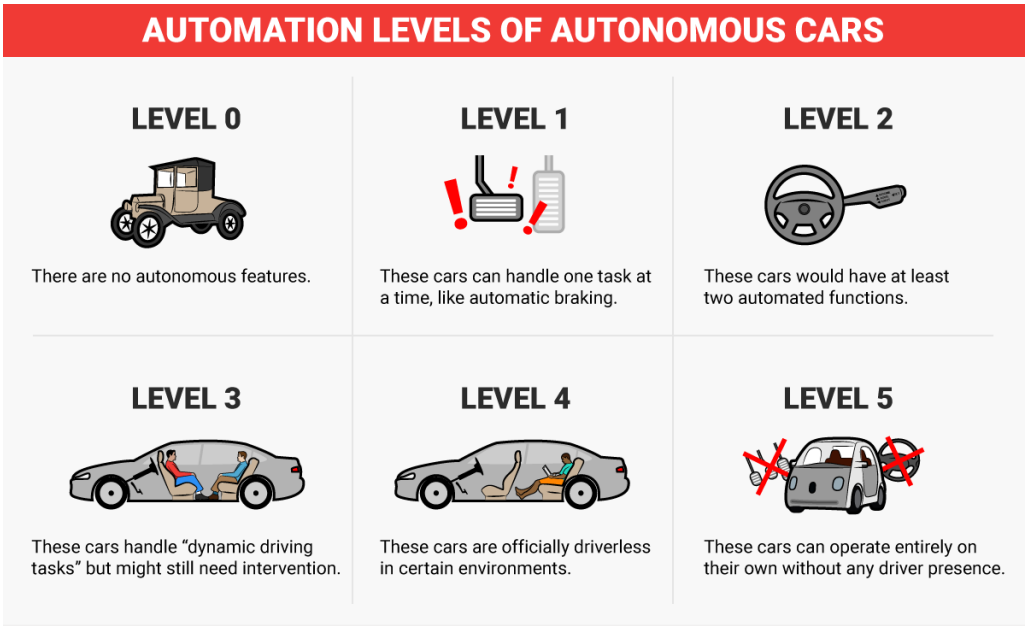

Figure 1: ADAS Levels

1.1 PolarFire SoC FPGA

Field-programmable gate arrays (FPGAs) are reconfigurable hardware devices. They can be reprogrammed to implement different combinational and sequential logic created with the aim of prototyping digital circuits, as they offer flexibility and speed. In recent years, technological advances have allowed the construction of FPGAs with considerable large amounts of processing power and memory storage. As so, they have been applied in several domains (telecommunications, robotics, pattern recognition tasks, infrastructure monitoring, etcetera).

Processors and FPGAs are the hardworking cores of most embedded systems. Integrating the high-level management functionality of processors and the stringent, real-time operations, extreme data processing, or interface functions of an FPGA into a single device forms an even more powerful embedded computing platform. SoC FPGA devices integrate both processor and FPGA architectures into a single device. Consequently, they provide higher integration, lower power, a smaller board size, and higher bandwidth communication between the processor and FPGA. They include peripherals, on-chip memory, an FPGA-style logic array, and high-speed transceivers[2]. Till now, most of the processors used in SOC FPGA were based on "Advanced RISC machine"(ARM).

The Polarfire SOC FPGA Icicle Kit (Figure [2]) used for this project is the world's first RISC V (an open-source architecture) based FPGA SOC. The board has a rich set of peripheral interfaces, including PCIe, 2 x Gigabit Ethernet, Micro USB, 4 x UART (via single micro-USB), 2 x CAN, 2 x SPI, 2 x I2C. The SOC consist of 5 core processor, a SiFive E51 Monitor core (RV64IMAC) and 4 SiFive U54 Application cores (RV64GC). The high-speed peripheral and the high processing power makes the board highly compatible with handling ADAS application.[3]

1.2 OpenCV

Image Processing is a form of signal processing in which the input is an image, such as a photograph or video frame, and the output is an image or set of characteristics

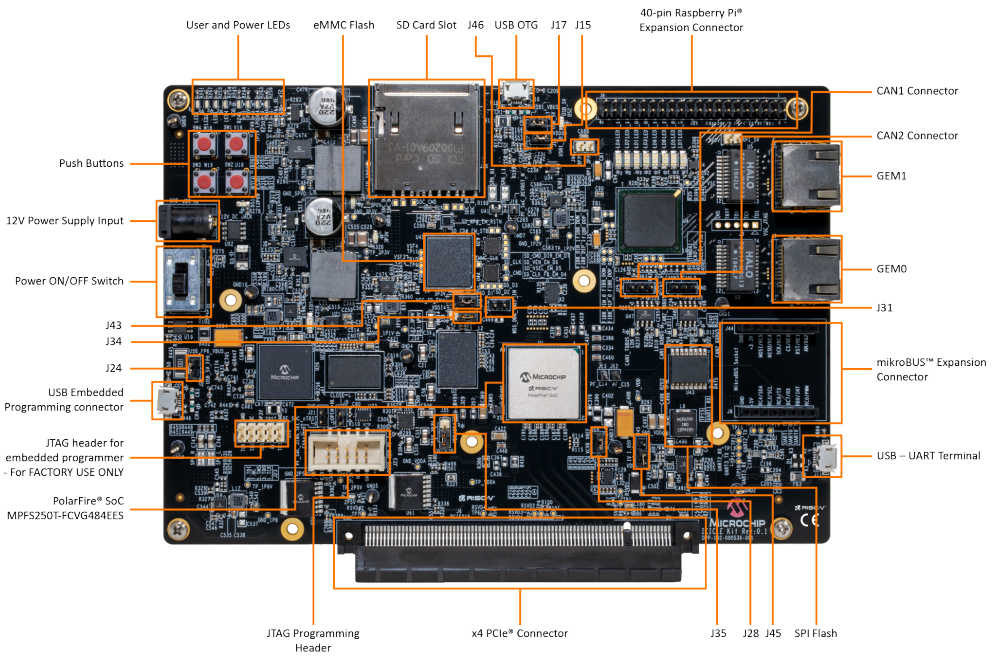

Figure 2: PolarFire SoC Icicle Kit

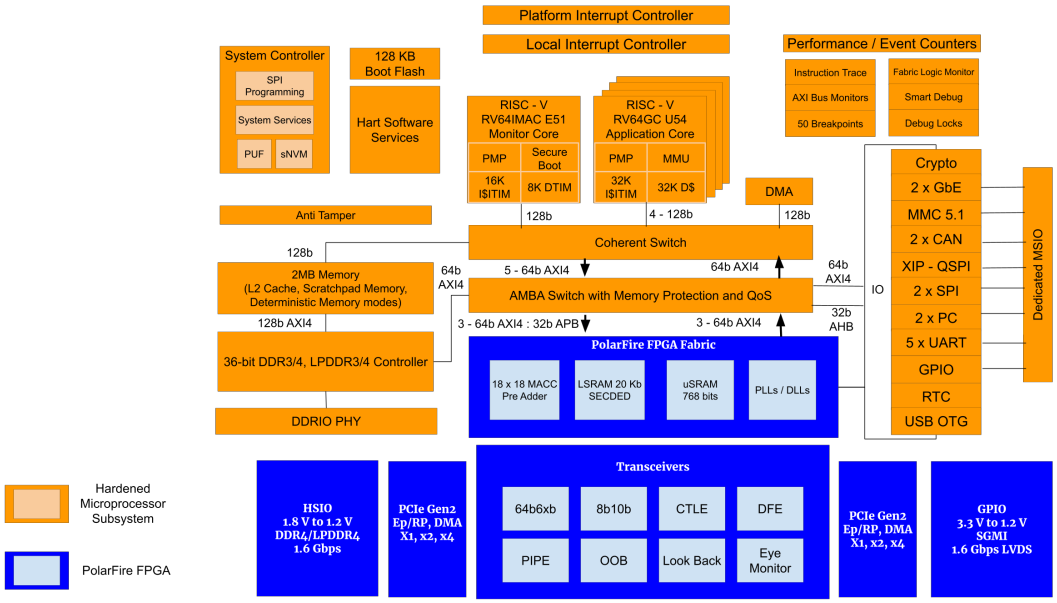

Figure 3: PolarFire SoC Block Diagram

related to the image. Open-Source Computer Vision Library (OpenCV) is an open-source computer vision and machine learning software library mainly used for image processing. It is freely available on the open-source Berkely Software Distribution license. It was started as a research project by Intel. OpenCV contains various tools to solve computer vision problems. It contains low-level image processing functions and high-level algorithms for face detection, feature matching and tracking. Some main image processing techniques are – Image Filtering, Image Transforming, Object Tracking, and Feature Detection[6].

Figure 4: OpenCV Canny Edge Detection

1.2.1 Canny Edge Detection

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. John F. Canny developed it in 1986.

Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. The general criteria for edge detection include the following:

1. Detection of edges with a low error rate, which means that the detection should accurately catch as many edges shown in the image as possible

2. The edge point detected by the operator should accurately localize on the edge's centre.

3. A given edge in the image should only be marked once; where possible, image noise should not create false edges.

To satisfy these requirements, Canny used the calculus of variations – a technique which finds the function which optimizes a given function. The sum of four exponential terms describes the optimal function in Canny's detector, but the first derivative of a Gaussian can approximate it.

The equation for a Gaussian filter kernel of size (2k+1)×(2k+1) is given by:

$$H_{ij} = \frac{1}{2\pi\sigma^2} exp(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}); 1 \leq i, j \leq (2k+1) \qquad (1)$$

The process of the Canny edge detection algorithm can be broken down into five different steps:

1. Apply a Gaussian filter to smooth the image in order to remove the noise
2. Find the intensity gradients of the image.
3. Apply gradient magnitude thresholding or lower bound cut-off suppression to get rid of spurious responses to edge detection
4. Apply double threshold to determine potential edges
5. Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges. [9]

1.2.2 Sobel Edge Detection

The Sobel operator, sometimes called the Sobel–Feldman operator or Sobel filter is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasising edges. Sobel and Feldman presented the idea of an "Isotropic 3×3 Image Gradient Operator" at a talk at SAIL in 1968. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel–Feldman operator is either the corresponding gradient vector or the norm of this vector. The Sobel–Feldman operator is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions. Therefore, relatively inexpensive in terms of computations. On the other hand, its gradient approximation is relatively crude, particularly for high-frequency variations in the image.

The operator uses two 3×3 kernels convolved with the original image to calculate approximations of the derivatives – one for horizontal changes and one for vertical. If we define A as the source image and G_x and G_y [Equation 2,3] are two images which at each point contain the horizontal and vertical derivative approximations, respectively, the computations are as follows: [10]

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A \tag{2}$$

and

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \tag{3}$$

Where $*$ here denotes the 2-dimensional signal processing convolution operation.

$$G = \sqrt{G_x^2 + G_y^2} \tag{4}$$

Using this information, the gradient's direction [Equation 4] can be calculated:

$$\theta = atan2(G_y, G_x) \tag{5}$$

Where, for example, θ is 0 for a vertical edge which is lighter on the right side.

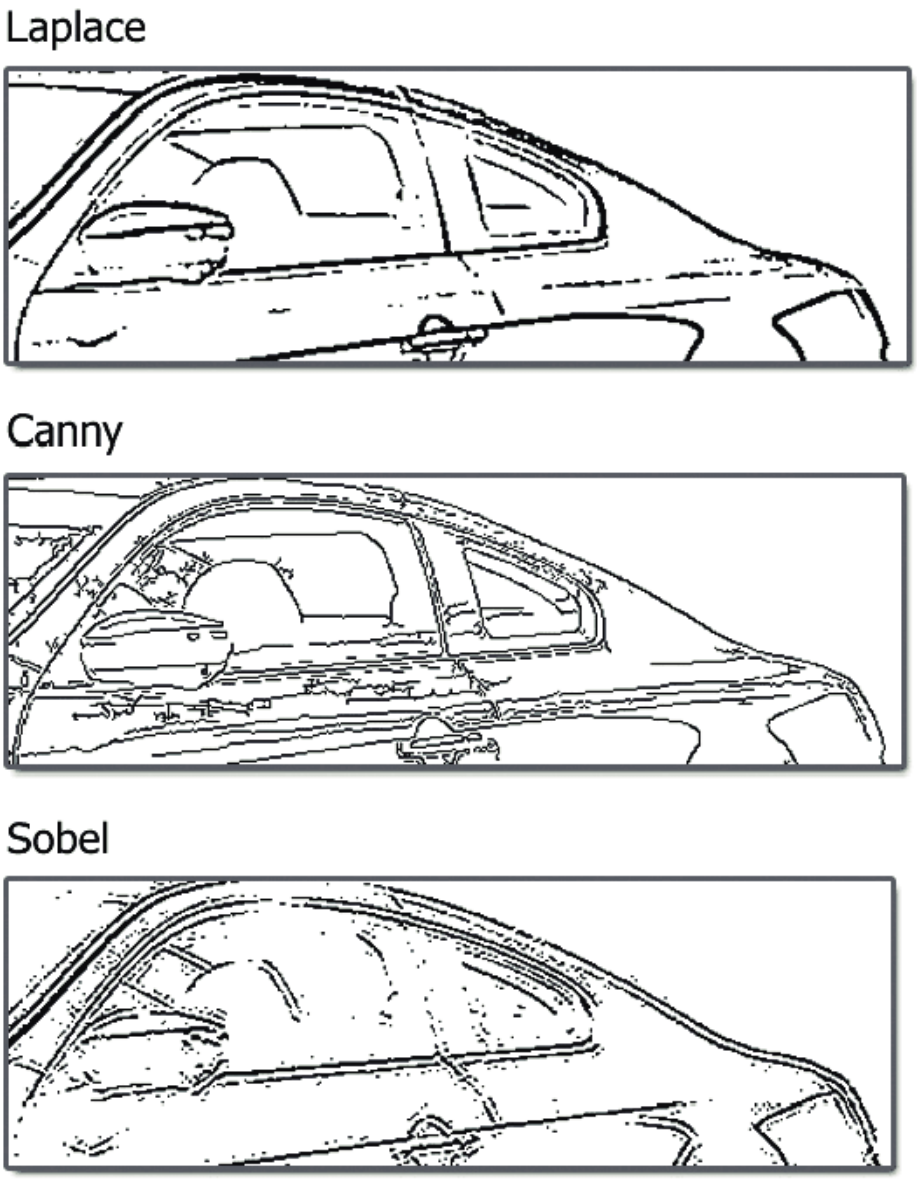

Figure 5: Edge Detection using OpenCV

2 Algorithm and Flow

So currently, we are looking at the lane detection of roads using the front-facing view from the car. This will be done on the hardware, i.e., the FPGA fabric, where the most computational process will happen, and the software, where the rest of the algorithm will run.

GitHub Repo Link

The algorithm is divided into three parts, Preprocessing, Detect Line, and Draw Lane, as shown in Figure 6. Some parts of the preprocessing will occur on the FPGA fabric. [Figure 15] [4]

The Preprocessing mainly highlights the edges of the lane so that the sliding window function can identify it. This includes using the Histogram equalizer func-

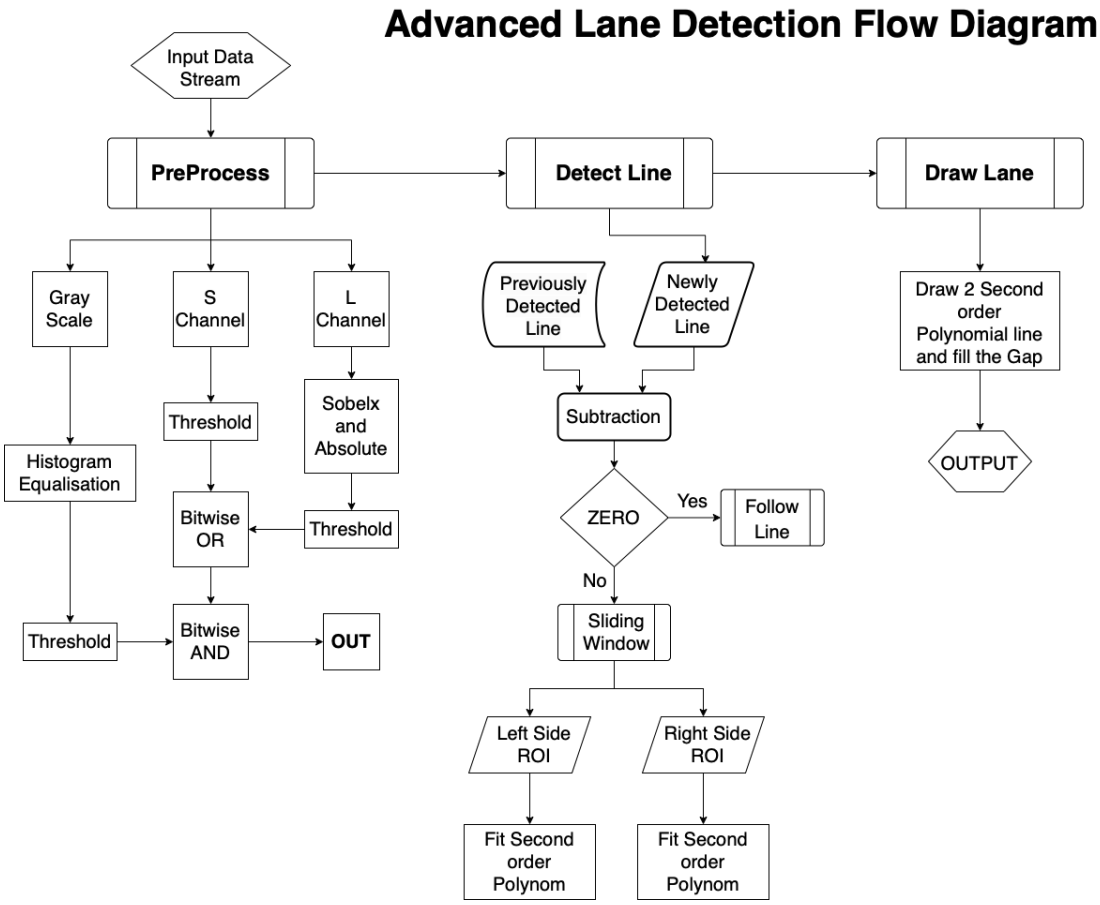

Figure 6: Complete Process Flow

tion and the *Sobel Operator* to get the edges. Histogram Equalization is a computer image processing technique that improves contrast in images. It accomplishes this by effectively spreading out the most frequent intensity values, i.e. stretching out the intensity range of the image. This method usually increases the global contrast of images when close contrast values represent its user data. This allows for areas of lower local contrast to gain a higher contrast. In our application, we are applying the function in the grayscale version of our image to increase the contrast ratio of the image. Then the processed image is combined with the image obtained from the Sobel operator using the *AND operation*.
The primary function of the Sobel Operator is to find the edges in the image, which is used to find the edges of the lane, which is thus used to find the lanes on the road. A very common operator for doing this is a Sobel Operator, which is an approximation to a derivative of an image. It is separate in the y and x directions. If we look at the x-direction, the Gradient of an image in the x-direction is equal to this operator. We use a kernel 3X3 matrix, one for each x and y direction. The Gradient for the x-direction has minus numbers on the left-hand side and positive numbers on the right-hand side, and we are preserving a small quantity of the centre pixels. For our use case, since we only have to get the edges of the lane on the vertical axis,in the image, we only have to use the Gradient in the X direction, which will give the vertical edges in our image. Hence using the 3x3 kernel, we perform the kernel convolution on the whole image to get the edges in it.

The above example shows the result of doing convolution by placing the Gradi-

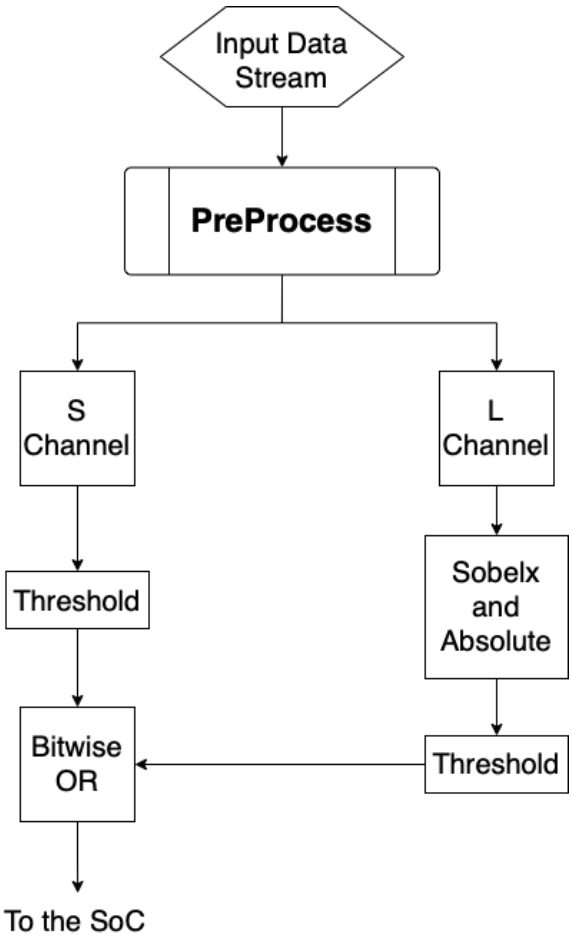

Figure 7: FPGA Implementation

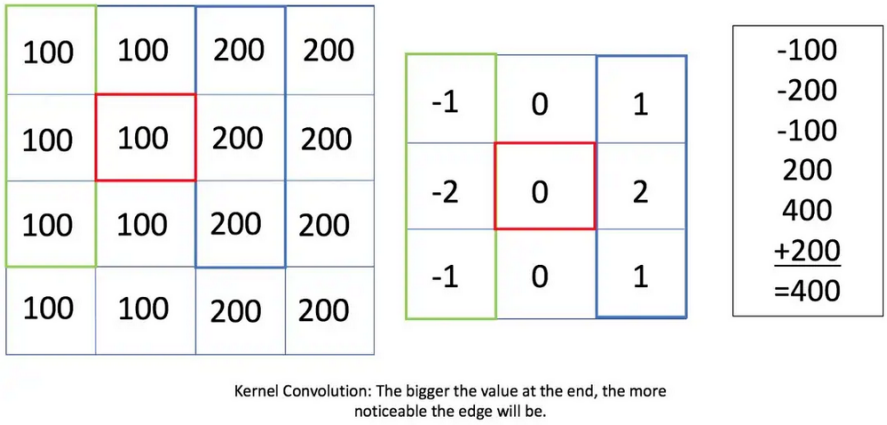

Figure 8: FPGA Implementation

ent matrix X over a red-marked 100 images. The calculation is shown on the right, which sums up to 400, which is non-zero. Hence there is an edge. If all the pixels of images were of the same value, then the convolution would result in a resultant sum of zero. So the gradient matrix will provide a significant response when one side is brighter. In our case, we are applying the Sobel operator on the Luminance of the image, and then we bitwise OR the output with the Saturation channel of the image. This output then goes to the output of the Histogram equalizer, where it gets bitwise AND with it.

The next step will be to take the perspective transform of the image in the top view

to apply the detect line procedure. This would help the algorithm to detect the lanes' edges quickly. This is the end of the algorithm preprocessing stage, and the next step will be to detect the lines of the lane, which is the Detect line Stage.

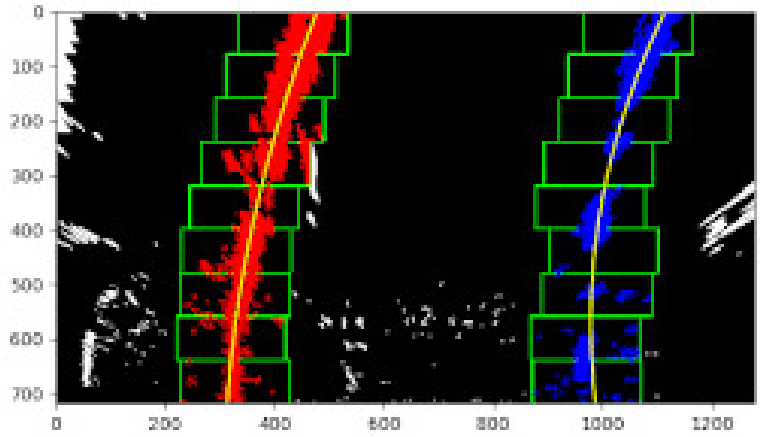

Figure 9: Sliding Window

We can express Lane's edge as a quadratic polynomial which can be found by calculating the indices of the pixels which are the lane's edge and using that to find the coefficients of a quadratic polynomial of the line. In the Preprocessing function, the two lines of the Strip have been tried to be as straightforward as possible. The line detection function changes according to whether or not there are stripes in the previous frames. The Line tracking function works if a lane has been found close in the previous frame. This function looks at the location of the lines belonging to the previously detected lane and compares it with lines in the current frame. If enough white pixel values are in and around these lines, it will follow the old found strip. The trailing windows algorithm works if there are not enough white pixels or no stripes in the previous frame. Otherwise, the Sliding Window function works to find the Strip in the new frame.
In the developed following windows algorithm, the frame is split into right and left parts from the middle of the y-axis. The histograms of the Right and Left parts are calculated on the x-axis, and the rectangular window is started from the maximum points of the histograms. The maximum point of the right or left histogram gives the starting point of the line of the Strip. Because with the preprocessing function, we aimed to eliminate the noise in the area we are interested in and other information other than line information. With the histogram obtained after a successful preprocessing function, We can find the Strip using the location on the peaks of the histogram.
Using the Starting point obtained from the histogram, we create a rectangular window with that as its centre point. The midpoint of the white pixels inside the first rectangular window drawn is taken tome follow the position of the line along the y-axis, and the next rectangular window is placed at this point centre and just above the previous window. This process is repeated iteratively. The coefficients of quadratic polynomial equations are found according to the midpoint of the drawn windows. These coefficients, found separately for the right and left frame parts, are equivalent to the mathematical expression describing the position of the right and left lines of the Strip. This will be the end of the Detect Line stage of the algorithm,

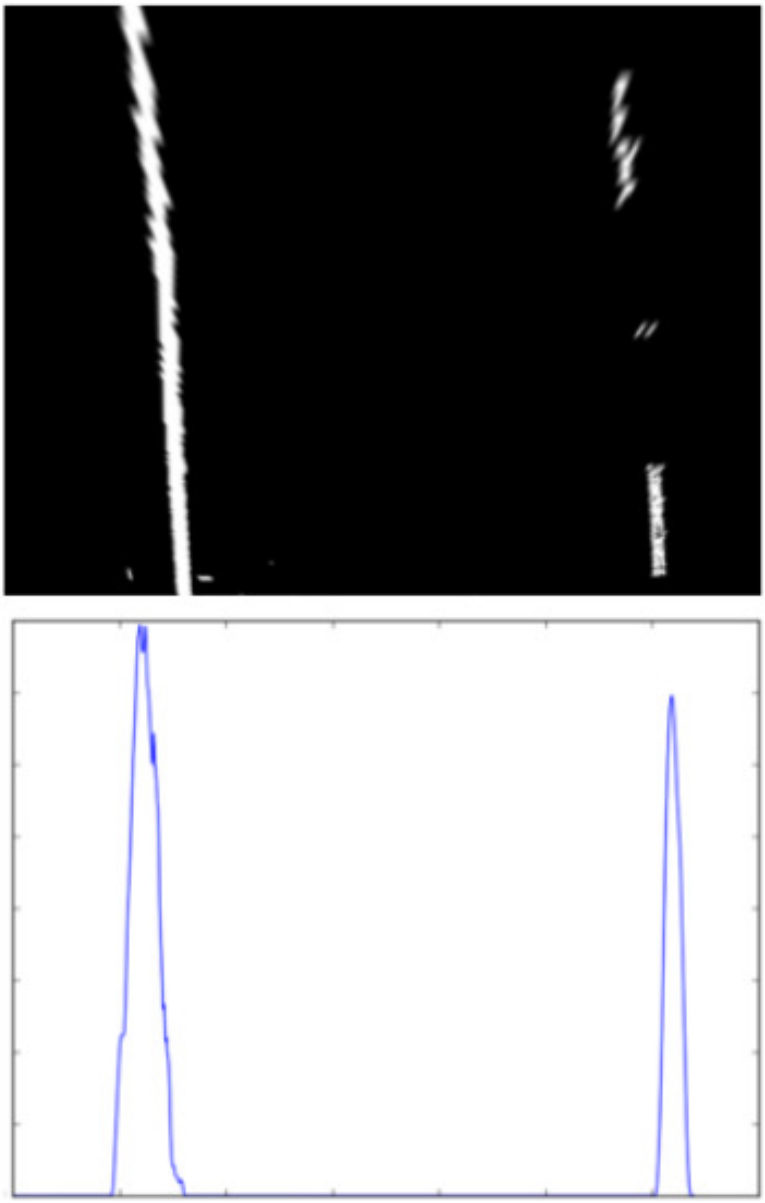

Figure 10: Histogram Visualization

and then we will go to the Draw Line Stage. In the Draw Line stage, using the coefficients of the polynomial, we can draw the Strip, which would be the lane's edges, and using OpenCV, we can highlight the lane by filling it with colour with both the strips acting as its border and thus the algorithm is complete. We have detected the lane from an image.

3 Results

The software time analysis for 480*360 resolution highway driving videos on a Lane Detection System are:

Based on the result in table 9 attempt has been made to accelerate the process using hardware acceleration provided by the fabric. The output of pre-processing is then taken as an input in rest of the processes of detection. The overall output will look as shown in figure 10. The Hardware Output is shown in Figure 11.

Functions	Number of test frame which is taken in a video	Maximum processing time	Minimum processing time	Average processing time
Pre-processing	1260	81.71 ms	8.21 ms	36.99 ms
All Functions	1260	153.31 ms	25.47 ms	77.84 ms
Pre-process/All	-	53.3%	32.2%	47.5%

Figure 11: Processing Time

Figure 12: Hardware Output

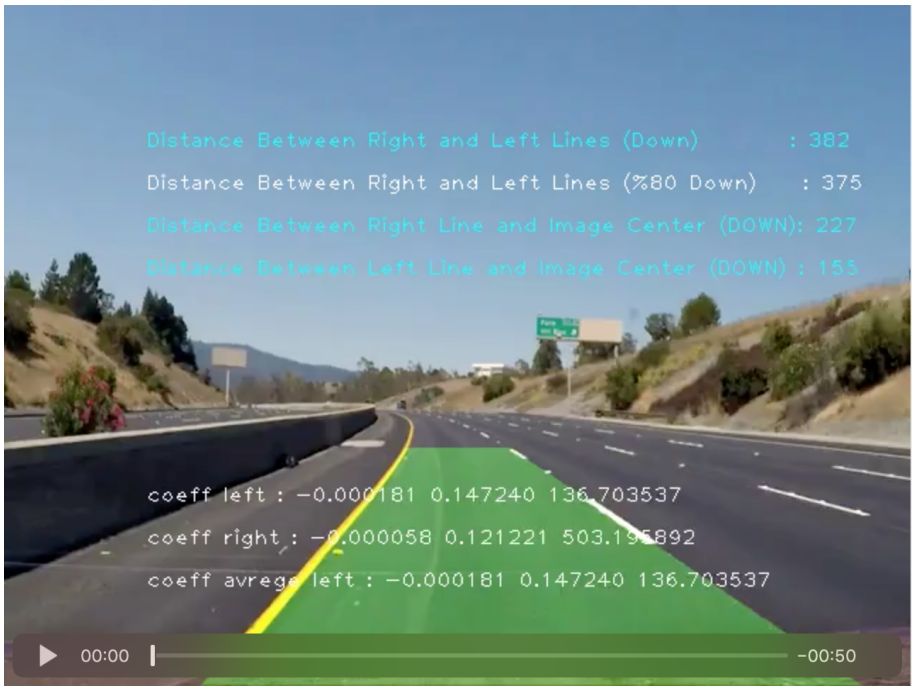

Figure 13: Final Output Image

We have also shown the hardware utilization of the FPGA's resourse in the Table in Figure 12 and 13.

Resource Usage

Type	Used	Total	Percentage
4LUT	30006	254196	11.80
DFF	40826	254196	16.06
I/O Register	0	432	0.00
User I/O	34	144	23.61
-- Single-ended I/O	34	144	23.61
-- Differential I/O Pairs	0	72	0.00
uSRAM	0	2352	0.00
LSRAM	768	812	94.58
Math	0	784	0.00
H-Chip Global	3	48	6.25
PLL	0	8	0.00
DLL	0	8	0.00
OSC_RC2MHZ	1	1	100.00
Transceiver Lanes	0	4	0.00
Transceiver PCIe	0	2	0.00
ICB_CLKINT	1	72	1.39
MSS	0	1	0.00

Figure 14: Resource Usage

4 Future Aspects and Conclusion

- It can detect exactly two lane lines, i.e. the left and right lane boundaries of the lane the vehicle is currently driving in.
- It cannot detect adjacent lane lines
- The vehicle must be within a lane and must be aligned along the direction of the lane
- If only one of the two-lane lines has been successfully detected, then the detection is considered invalid and will be discarded. In this case, the pipeline

Detailed Logic Resource Usage

Type	4LUT	DFF
Fabric Logic	2358	13178
uSRAM Interface Logic	0	0
LSRAM Interface Logic	27648	27648
Math Interface Logic	0	0
Total Used	30006	40826

Figure 15: Detailed Logic Usage

will instead output a lane line fit (for both left and right) based on the moving average of the previous detections. This is due to the need to implement the lane approximation function.

Above are the limitations of the project which we would like to consider for future work. Also, implementing Object Detection and Object Tracking algorithms to make this a complete ADAS system would be considered in future.

With the traditional Computer Vision approach, it can be learnt that the solutions developed through such approaches need to be optimised and can be sensitive to the chosen parameters. As a result, Deep Learning based approaches to Computer Vision would be better and accurate. Although they can appear as a black box at times, Deep learning approaches avoid the need for fine-tuning these parameters and are inherently more robust. [7]

The challenges encountered were almost exclusively due to non-uniform lighting conditions, shadows, discolouration and uneven road surfaces. Although it was not difficult to select the thresholding parameters to successfully filter the lane pixels, it was very time-consuming. Furthermore, the two biggest problems with the pipeline that becomes evident are:

1. Its inability to handle sharp turns and the constantly changing slope of the road. This is a direct consequence of the assumption made in Step 2 of the pipeline, where the road in front of the vehicle is assumed to be relatively flat and straight(ish). This results in a 'static' perspective transformation matrix, meaning that the lane lines will no longer be relatively parallel if the assumption does not hold. As a result, it becomes much harder to assess the validity of the detected lane lines because even if the lane lines in the warped image are not nearly parallel, they might still be valid.

2. Poor lane lines detection in glare/ severe shadows/combination of both. This results from the failure of the thresholding function to successfully filter out the lane pixels in these extreme lighting conditions

References

[1] How Machine Learning in Automotive makes Self-driving cars a Reality. https://mindy-support.com/news-post/how-machine-learning-in-automotive-makes-self-driving-cars-a-reality/, 2020.

[2] CORPORATION, A. What is an SoC FPGA? https://www.intel.com/content/dam/support/us/en/programmable/support-resources/bulk-container/pdfs/literature/ab/ab1-soc-fpga.pdf, 2014.

[3] CORPORATION, M. Polarfire SoC Icicle Kit Quick Start Guide. https://www.microsemi.com/products/fpga-soc/polarfire-soc-icicle-quick-start-guide, 2020.

[4] GÖRÜR, Y. Lane detection with it's implementation on fpga. https://github.com/ykpgrr/Lane-Detection-with-Implementation-on-FPGA, 2020.

[5] MOUJAHID, A., TANTAOUI, M. E., HINA, M. D., SOUKANE, A., ORTALDA, A., ELKHADIMI, A., AND RAMDANE-CHERIF, A. Machine learning techniques in adas: a review. In *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)* (2018), IEEE, pp. 235–242.

[6] NAVEENKUMAR, M., AND VADIVEL, A. Opencv for computer vision applications. In *Proceedings of national conference on big data and cloud computing (NCBDC'15)* (2015), pp. 52–56.

[7] NOORKHOKHAR. Advanced lane lines detection. https://github.com/noorkhokhar99/Advanced-Lane-Lines-Detection, 2019.

[8] OF INDIA, G. *Road Accidents in India 2021*. 2021.

[9] WIKIPEDIA. Canny operator. https://en.wikipedia.org/wiki/Canny_edge_detector.

[10] WIKIPEDIA. Sobel operator. https://en.wikipedia.org/wiki/Sobel_operator.