International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 14, No.1 (Sep-2023)

http://dx.doi.org/10.12785/ijcds/140185

FPGA based object parameter detection for Embedded Vision Application

Modi P R Prasad[1] and Arvind K Singh[2]

[1,2]*Department of Electrical Engineering National Institute of Technology, Kurukshetra, India*

Received 14 Jul. 2022, Revised 16 Sep. 2023, Accepted 19 Sep. 2023, Published 22 Sep. 2023

Abstract: Real-time image/video processing contains complex algorithms, and the number of computation operations such as a number of additions and the number of multiplications, which are solved by CPU, results decrease in processing speed and consume more power. On the other hand, FPGA has high processing speed and low power consumption and it also has the capability to work with CPU. Hence, sharing the load and performing complex tasks, results increase the system performance. In this work, there is the development of IPs for single object detection and canny edge detection using different filters. These IPs can be used in FPGA for real-time image processing. The tool which is used is Vivado HLS, to generate the hardware accelerator and the library which is used is the xfOpenCV library. Both are developed by Xilinx. XfOpenCV library is the optimized version of OpenCV means to reduce the complexity of an algorithm for fast performance. Single object detection detects the object in real-time by using FPGA based system and xfOpenCV APIs. Similarly, edge detection using a different filter shows the effect of filters, which means removing noise in the image and making it smooth, for edge detection. Smoothing image also reduces the number of computations (Addition and Multiplication). Resource utilized by the generated IPs is very less which makes the system less heavy and more reliable.

Keywords: Zybo board, Vivado HLS, Object Detection, xfOpenCV, Edge detection

1. INTRODUCTION

The role of FPGA-based object parameter detection for embedded vision applications is attracting a lot of research interest due to novel and innovative software and hardware implementation. Various image processing techniques are applied which treated the image as a two-dimensional signal. The main purpose of real-time image processing is to remove unwanted noise and disturbance from the image and improve the motion picture quality. Increasing picture/image quality increases the pixel's useful information related to its own image such as the corner pixel. Corner pixels in the image contain useful information and are used as a key point in many applications, such as moving object tracking. Some applications such as motion detection required previous data to compare with current data for detection. Storing previous data required memory along with being closer to the processing unit to make the system process fast.

However, processing video images with a more complicated algorithm requires a lot of computing power due to the high pixel count. On the other hand, FPGA can collaborate with the CPU to make it appropriate for image processing. For high-speed operation, FPGA makes use of a tackle accelerator. The FPGA performs several pre-processing tasks to speed up processing. FPGA, which has the ability to regulate the process without involving the CPU, can also be built with the control sense in place and enforced [1], [2].

FPGA is a programmable device that consists of a number of arrays of an element that can be programmed directly by the end-user. The user can develop or design their own image processing application in C/C++. And convert them into hardware descriptive language and then implement them on FPGA. The array element in FPGA is DSP slices, LUTs (lookup tables), and memory cells. These DSP (Digital signal processing) slices are used for the fast operation of image algorithms. LUTs are used for logical arithmetical operations. Memory cells are used to store the data, done by LUTs and again reuse the data whenever they are required. Along with hardware, optimized software code is also important. Therefore xfOpenCV which is an optimized computer vision library is used for effective Performance [3], [4].

The contributions of this work:

- The novelty of this work is a Software implementation of canny edge detection using xfOpenCV library on Vivado HLS tool targeted for Zynq7000 based Zybo board achieves better accuracy and better efficiency compared with other existing methods [5] using the proposed algorithm and in this implementation, flipflop resource usage is only 21% and lookup table resource usage is only 49%.

- The proposed FPGA implementation is cost-effective and experimental results achieve good results compared with the existing method [5].

The Canny edge detector is known for its optimality based on three criteria of good detection, good localization, and good response to an edge. Based on the three criteria of good detection, good localization, and good reaction to an edge, the Canny edge detector is renowned for its optimality. Because it incorporates Hysteresis, Thresholding, Non-Maximum Suppression, and Sobel Edge Detection in addition to Sobel Edge Detection, Canny Edge Detection produces the greatest results.
As a result, the algorithm's final phases can identify and link edges with greater flexibility. Most edges may be found using the clever edge detector. Sigma, the standard deviation for the Gaussian filter, and the threshold values have a significant impact on the performance of the canny method. The size of the Gaussian filter increases as the value increases. This suggests greater blurring, which is required for noisy images, as well as the detection of sharper edges. When one of the components, such as localization or detection, reaches infinity, the Canny optimization approach suffers a significant disadvantage. The maximization of the product in this situation does not ensure that the other factor is still reasonable. Boundary conditions for the suggested technique are pixels in the foreground (highest value) and pixels in the background (lowest values).
The shortcomings of the canny edge detector include its narrow focus on local alterations, lack of semantic segmentation, and low accuracy.
Any edges with intensity gradients more than the maximum value are regarded as edges, whereas any below the minimum value are not. According to the connectivity, those that are in between two thresholds are either edges or not.
The organization of the paper is as follows: Section 2 consists of an overview and literature survey, and methodology is explained in Section 3, followed by simulation results, and Section 4 is the conclusion and future scope, followed by references.

2. OVERVIEW AND LITERATURE SURVEY

A. *Overview Of Zynq-7000 Soc*

The Zynq-7000 family is integrated with a feature-rich single or dual-core ARM cortex-A9-based processing system (PS) and programmable logic (PL). Figure 1 shows an overview of the Zynq SoC architecture, with the right part being Processing System (PS) and the left being Programmable Logic (PL). The processing system consists of many components such as on-chip memory, external memory interface, DMA controller, Watchdog timer, and peripheral controller with I/O multiplexed to 54 dedicated pins (MIO pins)[6], [7].

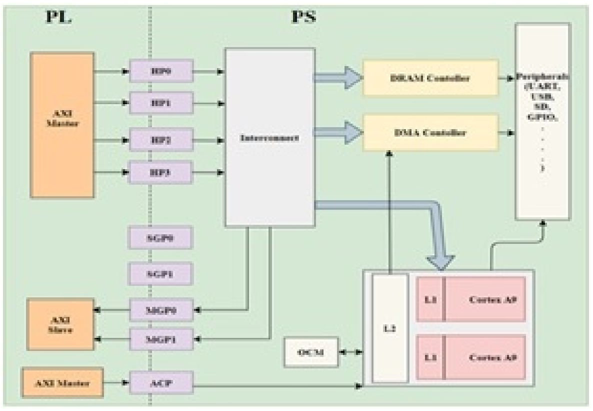


Figure 1. Overview of Zynq Architecture

B. *softwaretool Overview*

Vivado HLS tool is developed by Xilinx and is used to implement the code in the FPGA. The programming language used to configure the FPGA is Verilog or SystemVerilog, which is a hardware descriptive language. But sometimes, complex algorithms are very difficult to develop using hardware descriptive language when compared to any high-level language such as C/C++. As a result, this tool transforms languages like C/C++ in to an appropriate Verilog format that may be used to program FPGA. Once the designer completes its coding, it can be verified and tested using this tool. After converting code into RTL format, these tools generate the IP for the following RTL code, which can be used in flashing FPGA [8].
Vivado HLS tool 2019.2 is used for the experiment. The flow of the tool is shown in Figure 2 where before implementing or flashing the RTL code into FPGA, the RTL code must go through various steps in the tool including the writing of code, error checking of the code, and checking the functionality of the code. Writing the algorithm is done in C/C++ and after the end of our synthesis, this code is converted into RTL format. Vivado HLS contains a number of directive pragmas. These pragmas are used to optimize the C synthesis. Various combinations of debugging and testing can be obtained by using these different numbers of directives pragmas for the best-optimized solution of an application.
Firstly we have to write the top-level function according to the desired functionality. After that user run the simulation to check the correctness of the top-level function. The simulation uses a test-bench file that calls the top-level function. Error and warning after the simulation are shown in the error and warning window. Once the synthesis is done the file is ready for IP packaging which can be performed by RTL export from tool[9].

C. *Xfopencv Library Overview*

Xilinx develop the optimized xfOpenCV library for the FPGA device. These libraries are designed to work on a High-level synthesis development environment and SDx development environment. XfOpenCV libraries are similar

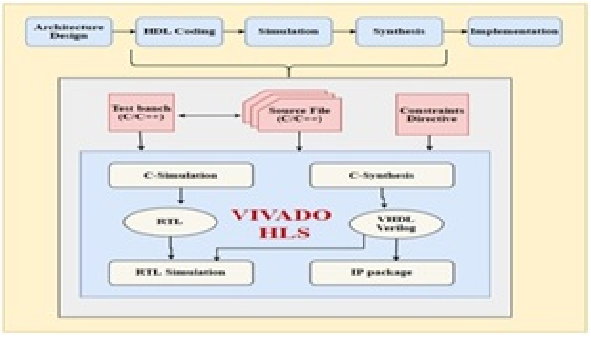


Figure 2. Workflow for Vivado HLS Synthesis

to OpenCV libraries in functionality. Both libraries are written in C++.
The major difference between xfOpenCV and OpenCV is that OpenCV is dedicated to runs on CPU and xfOpenCV is dedicated to runs on FPGA devices. As a result of which it gives up to 40x faster than GPUs and 90x faster than CPU.
Some classes and objects in the XfOpenCV library which are very useful in this project.

- xf:: Mat2AXIvideo class converts the image object from Mat format to AXI format.
- xf:: Canny function used for edge detection
- xf:: GaussianBlur function used for Gaussian filter
- xf:: medianBlur function used for bilateral filter
- xf:: bilateral filter function used for the median filter
- xf::threshold function used to remove the pixels from an image which are below the threshold level
- xf:: fast function used to detect the corner in the image.
- xf:: bounding-box used to make the rectangle box.

The common format used for using xfOpenCV:

- xf' namespace is used to define all functions.
- Templates are used to design the functions.
- Xf::Mat is used as an argument for an image.
- xf:: Mat2AXIvideo class converts the image object from Mat format to AXI format.
- xf:: Canny function used for edge detection
- xf:: GaussianBlur function used for Gaussian filter
- xf:: medianBlur function used for bilateral filter
- xf:: bilateral filter function used for the median filter
- xf::threshold function used to remove the pixels from an image that are below the threshold level
- xf:: fast function used to detect the corner in the image.

D. Single Object Detection Overview

Object detection plays an important role in real-time image processing like in an automatic electric vehicle, ship detection, security surveillance, and satellite-based detection. Real-time image processing required very fast processing such as in automatic electric vehicles to track the object present on the road should be detected very fast otherwise it results in a serious accident. Also in electric vehicles, power saving is an important factor. Therefore making a system that has fast processing and less power consumption is very useful. So, in this experiment, we are detecting a single object in an image using xfOpenCV libraries. XfOpenCV libraries have similar functionality as OpenCV. These libraries are optimized for Vivado HLS by Xilinx making the system fast. FPGA uses a hardware accelerator for high-speed operation and reduces the amount of work that is to be done by the CPU. Also, the power required for FPGA is very less which makes the system power efficient [10].

E. Edge Detection Using Different Filter Overview

Image processing is used in many applications. To get accurate and correct results from the application, different filters are used to reduce the noise in the image and also to smooth the image for fast processing. In this work, we are detecting the edge of an object using a canny edge algorithm and see how the edge detection [11] is affected when we use different filters such as the Gaussian filter, bilateral filter, and median filter. Filters are used to smooth the image and reduce the noise in the image which reduces the computation process (addition and multiplication). Canny edge and different filters are from xfOpenCV libraries which are optimized by Xilinx for the Vivado HLS tool [11]. FPGA is used for high-speed operation and low power consumption. We also see how much resource is utilized by a different filter in FPGA [12].

3. METHODOLOGY

A. Single Object Detection

1) Functions Used

a) Threshold

According to the specified value [5], this function adjusts the intensity of pixels to either minimum or maximum [5]. Threshold operation can be used for a wide variety of operations.

Step 1 The pixel intensity is set to the highest value if it is more than or equal to the specified threshold, else it is set to the minimum value.

Step 2 Whenever the pixel intensity crosses the set threshold, it is left as it was in the original picture pixel.
Step 3 Whenever the pixel intensity crosses the set threshold, it is set to zero (the minimum).
Step 4 The pixel intensity is set to a maximum or lowest in step 1 by the threshold and maximum values.
Step 5 In step 1, the output of the threshold picture is assumed to have two kinds of pixels: pixels with the highest value in the foreground and the lowest value in the background.

b) In range

This function removes the background pixels entirely from an image [5]. lower and upper values of boundaries are detected using this function. If it is false, the output pixel is set to 0 (dark/black), and if it is true, the output pixel is set to 255 (bright/white).

Output(Image)=Lower Value<Input (Image)<Upper Value

Following the identification of the foreground pixels in the image, the Fast (Features from accelerated segment test) corner detection method is carried out. The quick function locates an object's corners that are visible in our image.

c) Corner Detection

The Fast (Features from accelerated segment test) corner detection method is carried out after the foreground pixels in the image have been located. The quick function locates an object's corners that are visible in our image [5], Corner of an object is only deemed to exist if 9 consecutive pixels with the same intensity are detected to exceed the specified threshold [5]. The corner that was located has a marking of 255 (bright/white), otherwise, it is marked as 0 [5].

d) Finding Pixel Location

After obtaining an object's corners, it is necessary to locate each corner that was found, which can be done by using the XF BITSHIFT function [5]. Until the data transfer size, the XF BITSHIFT function counts each time the image size shifts to the right [5]. The x and y coordinates of any point where a value (pixel) is found are stored in the array when this function is used in a loop to check every row and column. The region of interest (ROI) determines which recorded coordinates are chosen [13]. An HSV image (HueSaturation-Value) is created initially from the color image. Take the HSV image's saturation feature out. Pixels that are lost during feature extraction are added back into the object using dilation [13]. The pixels are divided into maximum and minimum pixel intensities by the threshold. The minimal value pixel is completely removed from the image using In Range. The corner of an item is found via fast corner detection. Choose the region of interest (ROI) and locate the corner location. Utilizing the boundary function, draw a boundary over the object visible in our image using the region of interest [14], [15].

B. Implementation

1) Steps For Algorithm

a) Designer creates the top-level function for synthesis in C++:
b) Detection of objects.
c) Detection of ip is a synthesis wrapper function.
d) Use the HLS #pragma interface pragma directive. The syntax is HLS INTERFACE axis register both port = input.
The output port on both the HLS Interface's axis registers.
e) Develop a class of image objects using source code.
f) Develop Hue-Saturation-Value variable with pragma directive.

C. Edge Detection Using Different Filters

An integral part of object detection is edge detection. The algorithms below are implemented to perform the process of edge detection. The canny edge detection methodology is used as the base for edge detection with different filters being used for noise removal. The performance of the algorithm with each filter is studied and the best possible combination is implemented to obtain more flawless object detection. The procedure for edge detection followed is explained below.

1) Functions Used
a) Canny Edge

It is the most commonly used algorithm for edge detection. It is a multistage edge detection algorithm [16][17].

b) Pre-processing

This stage is required to remove or filter out the noise present in the image. Smoothing the image reduces the noise which is done by using a different filter such as a Gaussian filter, bilateral filter, and median filter.

2) Gradients

Gradient magnitude and direction are important parameters in canny edge detection. The algorithm calculates the gradient and finds out whether this single point in the image is possibly an edge or not. High magnitude means it is considered an edge and low magnitude means the point is not considered as an edge. The gradient's direction reveals information about the edge's orientation. The computation of the gradient's magnitude and direction (angle) comes next.

$$M(x,y) \Rightarrow \sqrt{{g_x}^2 + {g_y}^2} \qquad (1)$$

$$\alpha(x,y) \Rightarrow tan^{-1}\left[\frac{g_y}{g_x}\right] \qquad (2)$$

3) Non-maximum suppression

In this stage of edge detection, the algorithm finds out in which direction pixels are moving. There are 4 possible ways in which pixels move top to bottom, left to right, top left to bottom right(diagonally), top right to bottom left(diagonally) [18], [19]. After finding the direction, the algorithm finds out the maximum magnitude pixel by comparing it with neighboring pixels' magnitude present in the detected direction [20].

4) Edge tracing

We marked pixels as edges whose gradient magnitude is greater than the given threshold. Once we detect the pixels edge tracing is applied to draw the edges on the image [5].

a) Gaussian Filter

The Gaussian filter function blurs the input image. Gaussian filter is a non-linear low pass filter. Gaussian filtering uses a Gaussian kernel to convolve each point in the input image. The value of Gaussian kernel coefficients depends on sigma. The larger the value of sigma produces greater blurring [13].

b) Bi-Lateral Filter

The median filter works by going pixel-by-pixel across the image and replacing each pixel's value with the median of its surrounding pixels [21], [22]. The weighted average of the intensity values of adjacent pixels is used by the bilateral filter to replace each pixel's value. The bilateral filter is similar to the Gaussian filter. But the bi-lateral filter also uses sigma color and sigma space parameters for better filtering the image and reducing noise much more efficiently [23].

c) Median Filter

A nonlinear digital filter is the median filter. The median filter operates by traversing through the image pixel by pixel and then replacing each pixel value with the median value of its neighboring pixels[24] [9], [25].

D. Implementation

a) Software Tool

Vivado HLS is a Xilinx tool used to flash the FPGA. The programming language used in FPGA is Verilog, which is hardware descriptive language. But sometimes it is very difficult to develop an application writing Verilog when compare to another high-level language such as C/C++. Simulation and Synthesis can also be done by using Vivado HLS. Once the designer completes its coding, it can be verified and tested using this tool. For testing the top-level edge detection function, a test- bench is created After verification is done, the designer synthesizes the desired code and generates IP.

b) Libraries

XfOpenCV is Xilinx's computer vision library written in C++, mainly developed for Xilinx hardware. Some classes and objects in the XfOpenCV library which are very useful in this project.

- xf:: Mat class uses to define the image object in mat format
- xf:: AXIvideo2xfMat class converts the image object from AXI format to Mat format
- xf:: Mat2AXIvideo class converts the image object from Mat format to AXI format
- xf:: Canny class used for edge detection
- xf:: GaussianBlur class used for Gaussian filter xf:: medianBlur class used for bilateral filter
- x:: bilateral filter class used for the median filter

c) Steps for the Algorithm

- Wrapper function used to wrap the intellectual property (IP), means the input and the output of the function become the input and output port of generated IP.
- Inside the wrapper function

1) Pragma directives are used to interface inHLS
2) Define the image object class inside the wrapper function. This image object is used to store the image once the image is converted into a matrix format from AXI format. This stored image is passed to the top-level function.
3) HLS stream variable using pragma directive
4) Data flowpragma

- Inside the Top-Level function

Firstly, convert the input image into a grey image, then pass the grey image into the filter for smoothing the image. Smoothing reduces the noise in the image and helps in reducing unwanted edge detection. Different filter is used for smoothing such as Gaussian, bilateral and median filter. Finally, the filtered output is the pass to the canny edge for edge detection.

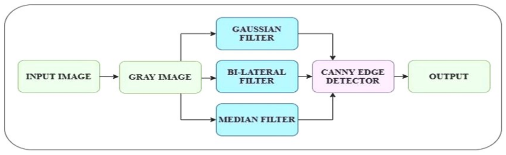


Figure 3. Flow of Algorithm in the proposed system.

E. Results

1) Simulation Results

Figure 4 is the Source image in Vivado HLS for C-simulation shown below. Figure 5-7 is the output image generated after the C-simulation is shown below respectively.

In Figure 5, the Gaussian filter reduces the noise present in

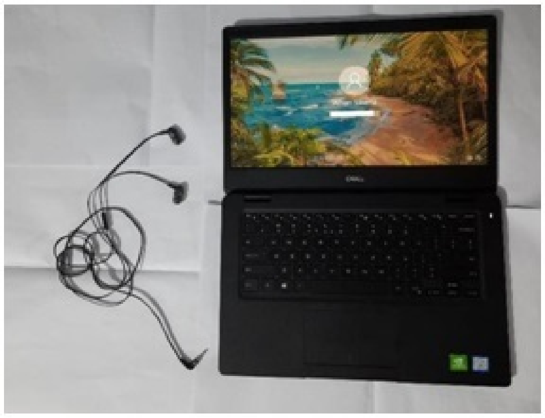


Figure 4. Input Considered.

the image as a result of which in edge detection very less unwanted edge is detected when compared to the median filter. In the median filter, noise is present which makes the edge detector detect an unwanted edge in the image. In Figure 5 and 7, the letter A and S in edge detection using a Gaussian filter contain less noise and produces fewer unwanted edge compared with a median filter. Also, a sharp edge is shown better in edge detection using a Gaussian filter rather than the edge detected by using a median filter.

Edge detection using a bilateral filter reduces noise much

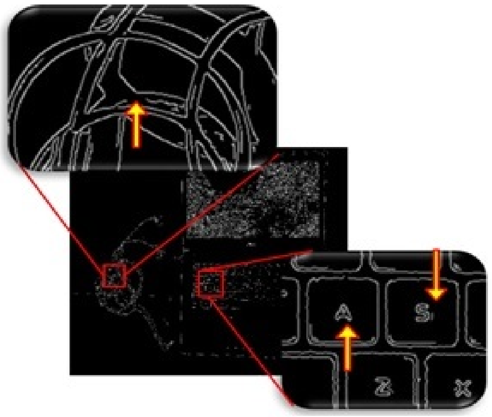


Figure 5. Canny using Gaussian.

better than Gaussian and median filters. Therefore the letter, A and S do not contain noise and no unwanted edge is detected. Edge sharpness is also better than the other two filters. In Figure 6, edge detection using bilateral catches

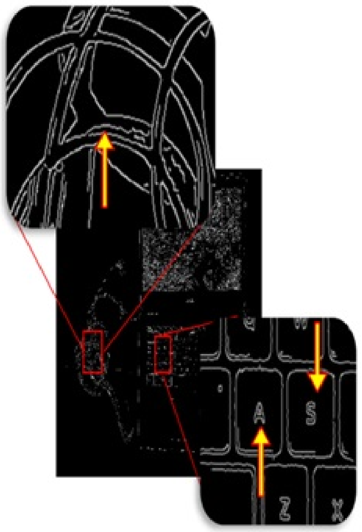


Figure 6. Bi-lateral Filter.

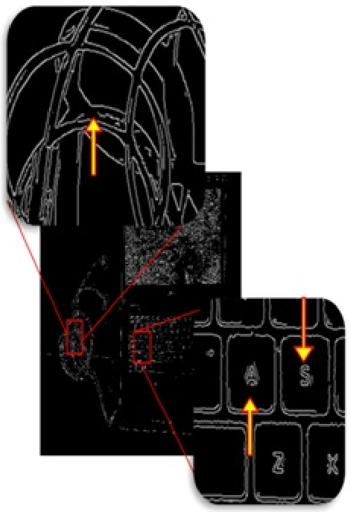
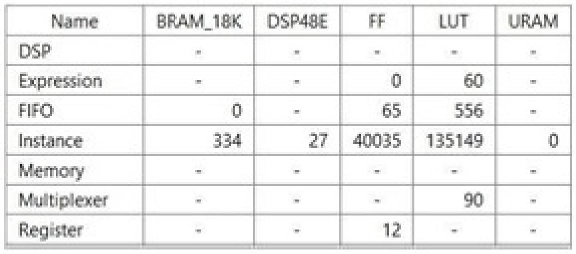


Figure 7. Median Filter.

the edge while using the other two filters fails to catch the edge as shown in Figures 5 and 6.

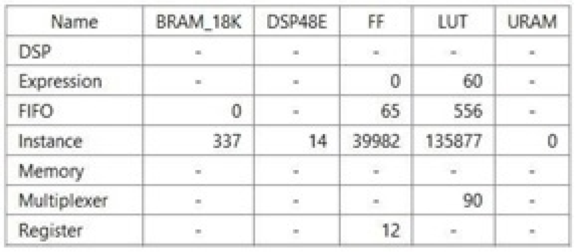
2) Resource Utilization

The Resource utilization when the canny algorithm is used with different filters is as follows. The results show the optimized resource utilization by FPGA after generating RTL in system design. Utilization of the lookup table (LUT) and the flip flop is nearly equal in all the edge detection [26], [27]. DSP(digital signal processing) uses is much more in edge detection using Gaussian filter and very less in median filter edge detection and moderate in bi-lateral filter edge detection [28], [29]. DSP helps in fast performance in filtering.



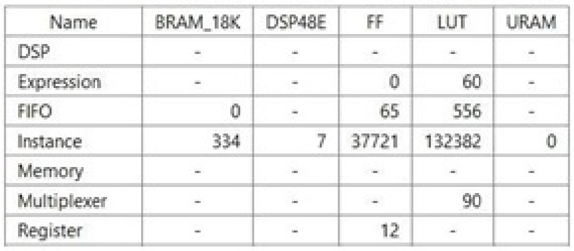
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	60	-
FIFO	0	-	65	556	-
Instance	334	27	40035	135149	0
Memory	-	-	-	-	-
Multiplexer	-	-	-	90	-
Register	-	-	12	-	-

TABLE I. RESOURCE UTILIZATION CANNY USING GAUSSIAN.



Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	60	-
FIFO	0	-	65	556	-
Instance	337	14	39982	135877	0
Memory	-	-	-	-	-
Multiplexer	-	-	-	90	-
Register	-	-	12	-	-

TABLE II. RESOURCE UTILIZATION CANNY USING BILATERAL.



Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	60	-
FIFO	0	-	65	556	-
Instance	334	7	37721	132382	0
Memory	-	-	-	-	-
Multiplexer	-	-	-	90	-
Register	-	-	12	-	-

TABLE III. RESOURCE UTILIZATION CANNY USING MEDIAN.

4. Final Results Of Single Object Detection

A. Simulation Result

The source and output images in Vivado HLS for C-Simulation are presented in Figures 8 and 9, respectively. The source image is processed by C-Simulation in Vivado in accordance with the xfOpenCV library function.
The results of the C- simulation is displayed in Figure 9. The detector object is contained in a rectangle in the output images.

B. IP GENERATED

Below is a screenshot of the created IP using Vivado HLS's export RTL function and C-synthesis.According to the AXI standard protocol, that contains AXI input and output ports.

C. Resource Utilization

The Vivado HLS utilization report, which displays the resources used by the FPGA to produce RTL during system design, is provided below. Only 1% of the flip flop is used in this approach, and only 3% of the lookup table, which is very little.

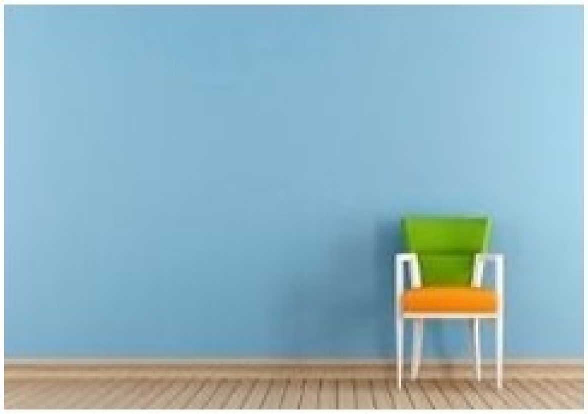


Figure 8. Source image in Vivado HLS.

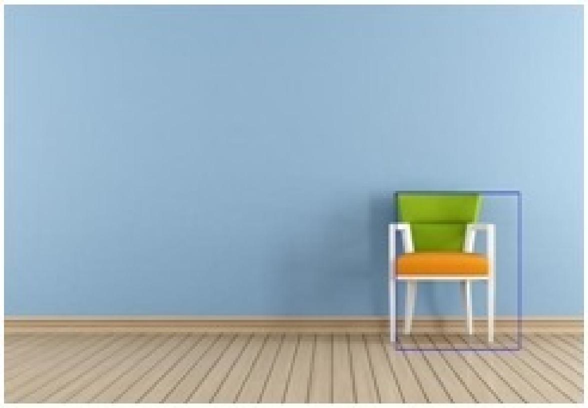


Figure 9. Output image after C-simulation.

5. Conclusions and Future Work

The xfOpenCV library in Vivado HLS has been successfully used for single object detection. Instead of being written in Verilog, complex image processing algorithms are written in high-level languages (C/C++). In comparison to the OpenCV library, the algorithm from the xfOpenCV library produces quicker results.
The resource utilization table demonstrates that the FPGA's resources are being used very sparingly. To achieve better outcomes, other filters can be applied throughout the procedure, such as the bilateral, median, and dilation filters. Compared to the OpenCV library, the approach from the xfOpenCV Library produces quicker results.
The result shows a comparison of the resource utilized by different filters. A bilateral filter reduces more noise as compared to Gaussian and median filters due to which a better edge is detected while using a bilateral filter. When compare to sharpness, edge detection using a bilateral filter gives better results than by using the other two filters. But if we compare between Gaussian and median filters, edge detection using the Gaussian filter gives good results than median filter edge detection. In the process of edge detection, selecting a threshold also plays an important role. By changing these value, the efficiency of smoothing

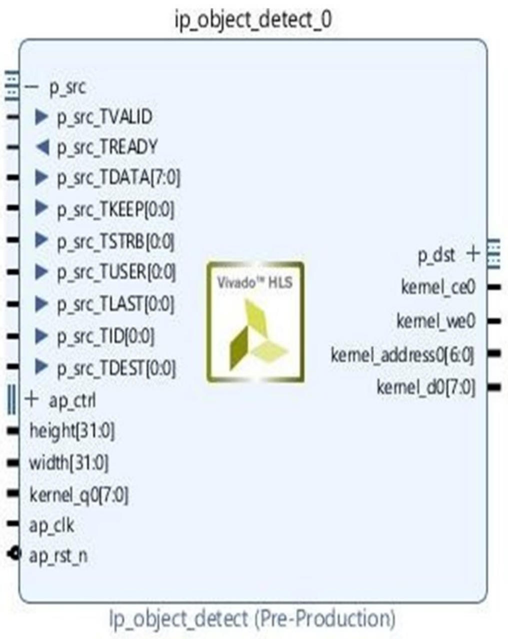
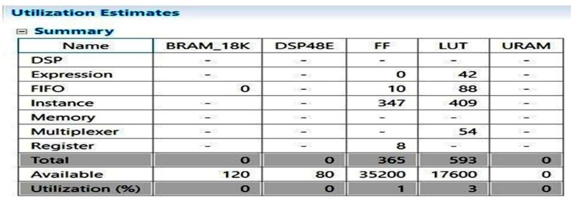


Figure 10. Generated IP.



Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	42	-
FIFO	0	-	10	88	-
Instance	-	-	347	409	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	54	-
Register	-	-	8	-	-
Total	0	0	365	593	0
Available	120	80	35200	17600	0
Utilization (%)	0	0	1	3	0

TABLE IV. UTILIZATION ESTIMATES.

the image can change which change the edge detection performance.
As in real-time image processing, performance is considered a very important factor. Better edge detection increases the accuracy to identify the object used in image processing applications such as security systems and object detection. It alters the image's smoothness, making it easier to identify an object's pixel placements. To prevent pixel loss of the targeted item, it is possible to alter the higher threshold and lower threshold values in the code. Changing the threshold improves the effectiveness of detection.
Real-time image processing plays an important role in Deep learning and machine learning application. But to resolve these algorithms, high processing speed and low-power consumption devices are needed. We can use powerful CPU's to fulfill our requirements but this will cost very high and also consume power. Alternate solution is FPGA, which has high performance and less power consumption. FPGA stands for field programmable gate array means it can be re-program whenever want. One special ability of an FPGA is to work with the CPU and change the complete system property and performance. The controlling part is done by the CPU and the complex algorithm part is done by FPGA, both parallel. 42 Many FPGA and Machine learning companies have collaborated to design a specific Tiny Embedded devices that can be capable of doing machine learning algorithms called TinyML. Embedded devices such as a smartwatch and cell-phones contain a small processor and limited battery power. To run these ML algorithms, force the device to consume more power and make the small processor heated. TinyML is the optimized machine learning algorithms mainly develop for small embedded devices.

References

[1] M. Ning, "A soc-based acceleration method for uav runway detection image pre-processing algorithm," in *2019 25th International Conference on Automation and Computing (ICAC)*. IEEE, 2019, pp. 1–6.

[2] A. B. Amara, E. Pissaloux, and M. Atri, "Sobel edge detection system design and integration on an fpga based hd video streaming architecture," in *2016 11th International Design & Test Symposium (IDT)*. IEEE, 2016, pp. 160–164.

[3] W. Liu, H. Chen, and M. Long, "Moving object detection and tracking based on zynq fpga and arm soc," in *IET Conference Proceedings*. The Institution of Engineering & Technology, 2015.

[4] S. Chhabra, H. Jain, and S. Saini, "Fpga based hardware implementation of automatic vehicle license plate detection system," in *2016 International Conference on advances in computing, communications and informatics (ICACCI)*. IEEE, 2016, pp. 1181–1187.

[5] M. K. Lokender Vashist, "Design of canny edge detection hardware accelerator using xfopencv library," *springer*, vol. 8, 2019.

[6] A. Cortes, I. Velez, and A. Irizar, "High level synthesis using vivado hls for zynq soc: Image processing case studies," in *2016 Conference on design of circuits and integrated systems (DCIS)*. IEEE, 2016, pp. 1–6.

[7] M. Kowalczyk, D. Przewlocka, and T. Kryjak, "Real-time implementation of contextual image processing operations for 4k video stream in zynq ultrascale+ mpsoc," in *2018 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. IEEE, 2018, pp. 37–42.

[8] X. Inc., *Xilinx OpenCV User Guide*, Xilinx Inc., 2021. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug1233-xilinx-opencv-user-guide.pdf

[9] T. Wu, W. Liu, and Y. Jin, "An end-to-end solution to autonomous driving based on xilinx fpga," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 427–430.

[10] Q. D. Ma, Z. Ma, C. Ji, K. Yin, T. Zhu, and C. Bian, "Artificial object edge detection based on enhanced canny algorithm for high-speed railway apparatus identification," *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pp. 1–6, 2017.

[11] Z. Fu, S. Song, X. Wang, J. Li, and H.-M. Tai, "Imaging the topology of grounding grids based on wavelet edge detection," *IEEE Transactions on Magnetics*, vol. 54, no. 4, pp. 1–8, 2018.

[12] M. Mittal, A. Verma, I. Kaur, B. Kaur, M. Sharma, L. M. Goyal, S. Roy, and T.-H. Kim, "An efficient edge detection approach to

provide better edge connectivity for image analysis," *IEEE Access*, vol. 7, pp. 33 240–33 255, 2019.

[13] E. Dong, B. Han, X. Yu, and S. Du, "Moving targets detection based on improved single gaussian background model," *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 1179–1183, 2018.

[14] M. Chen and P. Liu, "A deep learning-based fpga function block detection method with bitstream to image transformation," *IEEE Access*, vol. 9, pp. 99 794–99 804, 2021.

[15] X. Zhang, L. Zhang, and X. Lou, "A raw image-based end-to-end object detection accelerator using hog features," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 1, pp. 322–333, 2022.

[16] M. Kalbasi and H. Nikmehr, "Noise-robust, reconfigurable canny edge detection and its hardware realization," *IEEE Access*, vol. 8, 2020.

[17] H. T. . Lee and J. Park, "Energy efficient canny edge detector for advanced mobile vision applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, 2018.

[18] S. Kim, S. Na, B. Y. Kong, J. Choi, and I.-C. Park, "Real-time ssdlite object detection on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 6, pp. 1192–1205, 2021.

[19] V. Malviya, I. Mukherjee, and S. Tallur, "Edge-compatible convolutional autoencoder implemented on fpga for anomaly detection in vibration condition-based monitoring," *IEEE Sensors Letters*, vol. 6, no. 4, pp. 1–4, 2022.

[20] V. Raghavendra and L. Shrinivasan, "Time efficient design and fpga implementation of distributed canny edge detector algorithm," in *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. IEEE, 2018, pp. 2135–2139.

[21] X. Wang, Y. Song, F. Hou, M. Zhang, A. G. Richardson, T. H. Lucas, and J. V. d. Spiegel, "Design of a real-time movement decomposition-based rodent tracker and behavioral analyzer based on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 9, pp. 1133–1143, 2022.

[22] W. He, J. Zhang, Y. Lin, X. Zhou, P. Li, L. Liu, N. Wu, and C. Shi, "A low-cost high-speed object tracking vlsi system based on unified textural and dynamic compressive features," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 3, pp. 1013–1017, 2021.

[23] R. Manoranjitham and P. Deepa, "Novel interest point detector using bilateral-harris corner method," *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pp. 1–4, 2017.

[24] S. Vishaga and S. L. Das, "A survey on switching median filters for impulse noise removal," *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]*, pp. 1–6, 2015.

[25] X. Fan, G. Xie, Z. Huang, W. Cao, and L. Wang, "Acceleration of rotated object detection on fpga," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 4, pp. 2296–2300, 2022.

[26] W. Xu, F. Li, Y. Jiang, A. Yong, X. He, P. Wang, and J. Cheng, "Improving extreme low-bit quantization with soft threshold," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 33, no. 4, pp. 1549–1563, 2023.

[27] G. Lee, J. K. Park, and J. T. Kim, "Ocdma codeword switching technique to avoid interference of time-of-flight lidar system for autonomous vehicles," *IEEE Sensors Journal*, vol. 23, no. 3, pp. 3090–3102, 2023.

[28] Y. Liu, J. Li, K. Huang, X. Li, X. Qi, L. Chang, Y. Long, and J. Zhou, "Mobilesp: An fpga-based real-time keypoint extraction hardware accelerator for mobile vslam," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 12, pp. 4919–4929, 2022.

[29] M. Xu, Z. Zhang, H. Li, Q. Luo, R. Dou, L. Liu, J. Liu, and N. Wu, "Hierarchical parallel vision processor for high-speed ship detection," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 3, pp. 1164–1168, 2023.

Modi P R Prasad presently working as an Asst. Professor in the Dept. of Electrical Engg. at National Institute of Technology Kurukshetra. India.
His research interests are Marine Robotic Vehicle Control, Renewable Energy, Electronic Instrumentation. He is a member of IEEE.

Arvind K Singh working as a Research Scholar in the Dept. of Electrical Engg. at the National Institute of Technology Kurukshetra. India.
His research interests are Power Electronics, Renewable Energy, & Electronic Instrumentation. He obtained his M.Tech in Power Electronics from NIT Kurukshetra.