International Journal of Engineering Insights: (2023) Vol. 1, Nro.1, Regular Paper
https://doi.org/10.61961/injei.v1i1.3

Fast FPGA-Based Image Feature Extraction for Data Fusion in Autonomous Vehicles.

Jeremias Gaia · Eugenio Orosco · Francisco Rossomando · Carlos Soria

Received: 14 May 2023 / Accepted: 10 August 2023

Abstract: Computer vision plays a critical role in many applications, particularly in the domain of autonomous vehicles. To achieve high-level image processing tasks such as image classification and object tracking, it is essential to extract low-level features from the image data. However, in order to integrate these compute-intensive tasks into a control loop, they need to be completed with the highest speed possible. This paper presents a novel FPGA-based system for fast and accurate image feature extraction, specifically designed to meet the constraints of data fusion in autonomous vehicles. The system computes a set of generic statistical image features, including contrast, homogeneity, and entropy, and is implemented on two Xilinx FPGA platforms - an Alveo U200 Data Center Accelerator Card and a Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit. Experimental results show that the proposed system achieves high-speed image feature extraction with low latency, making it well-suited for use in autonomous vehicle systems that require real-time image processing. Moreover, this system can be easily expanded to extract extra features for diverse image and data fusion applications

Keywords FPGA · SoC · Image Processing · xfOpenCV

1 Introduction

Images are used as inputs in a variety of systems [1, 2,3] , and techniques for extracting information from them are the foundation of many applications such as simultaneous localization and mapping (SLAM) [4], autonomous driving safety [5], and human-robot interaction [6]. However, achieving high-speed image processing for these applications is computationally expensive for standard CPUs due to time constraints and large image sizes. To address this challenge, Field Programmable Gate Arrays (FPGAs) can generate dedicated hardware with low latency and high throughput computation[7], making them a suitable means of assisting general-purpose CPUs. FPGA technology can be seen as an additional layer of the system that performs parallelizable tasks, thereby reducing the CPU's computational load and, as a result, latency [8].

FPGAs are capable of simultaneously handling multiple standard communication ports, such as CAN bus, Ethernet, USB, etc., at high speeds. In the context of data fusion, the availability of such a device enables the creation of a communication-centric platform between connected sensors [9]. Additionally, by utilizing parallel structures, complex calculations can be performed significantly faster.

This paper presents a high-speed FPGA-based implementation for computing a set of common statistical image features that is performance-optimized by exploiting FPGAs' capacity to integrate sequential and parallel processing. The main contributions of this paper include a design that performs feature extraction with only one pass of the image through the system, and a primary design restriction of low latency suitable for operation inside a fast data fusion algorithm.

The rest of the paper is organized as follows: Section 2 contains related research articles, expressions used to determine different statistical image features are presented in Section 3, and Section 4.1 describes the proposed FPGA-based system. Experimentation results are shown in Section 4.2, followed by the conclusions in Section 5.

2 Related work

In conventional image processing applications, software developers commonly utilize the recognized OpenCV library [10], which uses the large amount of hardware resources available on computers to perform its oper-

Jeremias Gaia · Eugenio Orosco
Francisco Rossomando · Carlos Soria
Instituto de Automática
Universidad Nacional de San Juan
San Juan, Argentina
Tel.: +54 264 4213303
{jgaia,eorosco,frosoma,csoria}@inaut.unsj.edu.ar

ations. However, when it comes to embedded systems where resources are limited, these operations need to be optimized. To address this issue, researchers have developed numerous algorithms to optimize image processing operations in FPGAs, such as color to gray-level conversion [11], histogram construction [12] and edge detection [13], among others.

Histograms are the first layer of statistical information that can be extracted directly from grayscale image pixels. Younis *et. al.* proposed in [12] a histogram construction hardware implementation. Their system received grayscale images (pre-processed with MATLAB) as input for a finite state machine to perform the histogram calculation.

Edge detection algorithms enable computer vision systems to detect objects or patterns present in the input image. Tsiktsiris *et. al.* [13] designed a fast edge detection module based on Sobel operator algorithm that outputs an image containing only the edges of the input. Recently, filtering techniques for event-based cameras such as a background activity filter and a mask filter, were also implemented on FPGA [14].

Similar to our work, Siddiqui *et. al.* [15] proposed an FPGA-based soft processor called Image Processing Processor (IPPro) for general purpose image processing. The authors designed and tested an instantiable component, since they focus on multi-core operation. However, the source code is not available.

The computation of Graylevel Co-occurrence Matrix (GLCM) has been explored in prior research, as seen in [16]. In this method, the authors propose partitioning the input image into 128 by 128-pixel patches to form the GLCM matrices. In a more recent study [17], researchers introduced a novel technique using a circular buffer unit per orientation to compute the co-occurrence matrix for each orientation.

Hardware description languages use the Register-transfer-level (RTL) abstraction to create high-level circuit representations, which can be used to derive lower-level representations and actual wiring [18]. Recently, Xilinx company has developed the xfOpenCV library [19], which provides a set of performance-optimized kernels for Xilinx FPGAs and SoCs that enable the translation of common computer vision operations from sequential software processing to parallel hardware processing. By utilizing these kernels, programmers can avoid dealing with the internal connections of the system since the library generates RTL code automatically.

3 Materials and Methods

In this section, the theoretical background used to determine different statistical image features is presented.

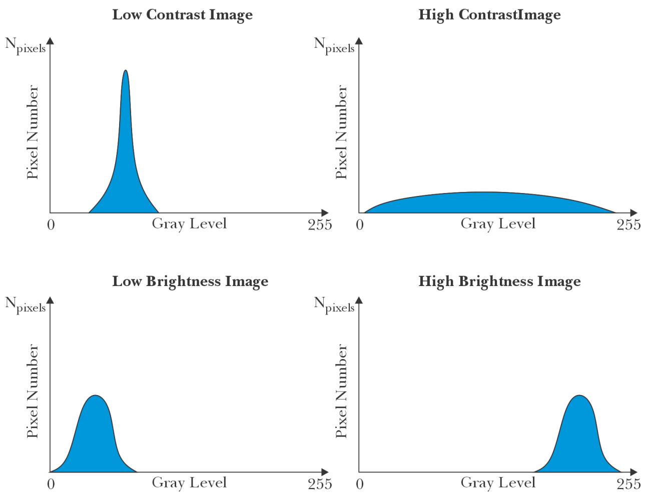

Fig. 1 A broad histogram denotes a high contrast picture since the image's pixels cover the entire range of gray levels, in contrast to a narrow and peaky histogram that indicates a low contrast environment.

3.1 Histogram-based measures

Histograms offer a lot of information about the image. Take as example the different histogram variations presented in Fig. 1. A broad histogram denotes a high contrast picture since the image's pixels cover the entire range of gray levels, in contrast to a narrow and peaky histogram that indicates a low contrast environment. Therefore, histogram-based measurements enable the quantification of the overall attributes of the image. Three histogram-based metrics are employed in this article: *histogram flatness measure (HFM), histogram spread (HS), and global entropy.*

3.1.1 Histogram Flatness and Histogram Spread

Introduced by Tripathi *et. al.* in [20], these metrics are very useful to assess image contrast. HFM can be expressed as the ratio of the geometric mean of the image histogram $h(x)$ to the arithmetic mean of $h(x)$ (Eq. (1)).

$$HFM = \frac{\left\{\prod_{i=1}^{n} x_i\right\}^{\frac{1}{n}}}{\frac{1}{n}\sum_{i=1}^{n} x_i} \tag{1}$$

where x_i is the count for the i-*th* histogram bin and n the number of histogram bins.

Histogram spread is the ratio of the inter-quartile distance to the range of the histogram (Eq. (2)). Here, the inter-quartile distance is the difference between the $3rd$ and the $1st$ quartile of the cumulative histogram. The $1st$ and $3rd$ quartile means the histogram bins at which cumulative histogram has 25% and 75% of the maximum value respectively.

$$HS = \frac{3^{rd}quartile - 1^{st}quartile}{max(h(x)) - min(h(x))} \quad (2)$$

3.1.2 Image Entropy

The global entropy of an image (Eq. (3)) measures the amount of low-level information contained in it. The probability for a pixel taking certain value in an image can be represented as the bin count for the given gray tone i in the image histogram h. Then, the global entropy can be defined as follows:

$$entropy = \sum_{i=0}^{255} h(i)\log_2(h(i)) \quad (3)$$

3.2 Gradient-based measures

Extracting the image gradient in a given direction is the first step for textural analysis. Image classification, edge enhancing, and visual feature extraction are some of the applications that rely in this principle. Therefore, being able to objectively evaluate how much "sharp" an image is, could help gradient-dependent algorithms to improve their performance.

3.2.1 Gradient Magnitude

An initial technique to assess image sharpness involves deriving the mean value from its magnitude image. Let G_x be the gradient image in the x direction and G_y the gradient image in the y direction, the magnitude image (G) can be defined as:

$$G = \sqrt{(G_x)^2 + (G_y)^2} \quad (4)$$

then, the mean value of G is

$$GradMagnitude = \frac{1}{rows * cols}\sum_{i=1}^{rows}\sum_{j=1}^{cols} G(i,j). \quad (5)$$

Greater values of the gradient magnitude measure indicate the presence of greater textural information in the image.

3.2.2 Gamma Sharpness (γ_{sharp})

Shin *et.al.* [21] proposed to apply the mapping function in Eq. (6) to the magnitude image G.

$$\hat{g_i} = \begin{cases} \frac{1}{N_g} log(\lambda(g_i - \gamma) + 1), & \text{for } g_i \geq \gamma, \\ 0, & \text{for } g_i < \gamma, \end{cases} \quad (6)$$

where $N_g = log(\lambda(1-\gamma)+1)$ is the normalization factor, g_i denotes the gradient magnitude at pixel i, γ indicates the activation threshold value for the mapping function, λ is a control parameter to adjust the mapping behaviour and $\hat{g_i}$ stands for the amount of gradient information at pixel i. Note that $g_i \in [0,1]$, assuming a pixel range of $[0,255]$ and a normalization factor of $1/255$. After the estimation of g_i, the γ sharpness measure can be obtained with Eq (7):

$$\gamma_{sharp} = \sum_{i \in G} \hat{g_i} \quad (7)$$

3.3 Haralick Textural Features

In 1973, Robert Haralick proposed a new approach to extract textural information from an image: the GLCM [22]. According to this method, the spatial relationship between the gray tones in an image I encodes the texture information for that image.

The first step to extract this information is to create the GLCM. All gray tone transitions in a given direction are counted and arranged in a MxM matrix, being M the amount of gray levels present in the image.

Consider for instance a digital image with pixel values ranging from 0 to 255. If all pixel values are present, the GLCM could be as big as 255 x 255. Once the GLCM has been computed, a set of features can be obtained by manipulating this matrix. We refer the reader to [22] for further details.

This work considers four textural features: *contrast, homogeneity, entropy and the information measure of correlation.*

In order to further understand the following equations, notation is provided.

- P refers to the graylevel co-ocurrence matrix.
- $p(i,j)$ is the (i,j)th entry in the normalized graylevel co-ocurrence matrix.

  $$p(i,j) = P(i,j)/R, \quad (8)$$

  being R a normalization factor.
- $p_x(i)$ refers to the ith entry in the marginal-probability matrix obtained by summing the rows of the co-ocurrence matrix P.

3.3.1 Angular Second Moment (energy)

$$f_1 = \sum_{i}^{N_g}\sum_{j}^{N_g}\{p(i,j)\}^2 \quad (9)$$

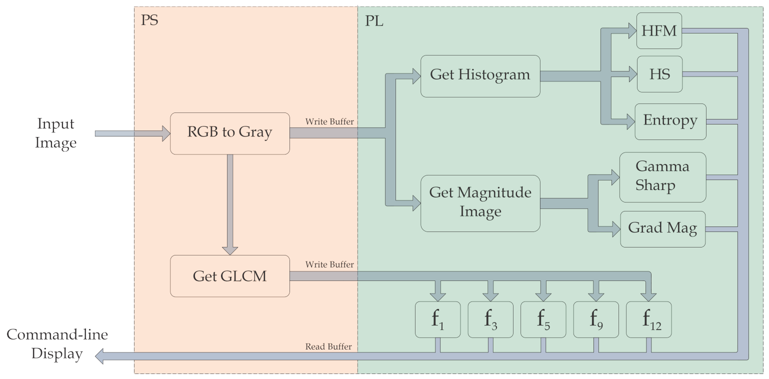

Fig. 2 Overview of the proposed system based on the Zynq UltraScale+ MPSoC ZCU104 architecture.

3.3.2 Contrast

$$f_3 = \sum_{i}^{N_g-1} n^2 \left\{ \sum_{|i-j|=n} \sum_{i}^{N_g} \sum_{j}^{N_g} p(i,j) \right\} \tag{10}$$

3.3.3 Homogeneity

Also known as inverse difference moment (IMC)

$$f_5 = \sum_{i}^{N_g} \sum_{j}^{N_g} \frac{1}{1+(i-j)^2} p(i,j) \tag{11}$$

3.3.4 Entropy

$$f_9 = -\sum_{i}^{N_g} \sum_{j}^{N_g} p(i,j) \log(p(i,j)) \tag{12}$$

3.3.5 Information Measure of Correlation (IMC)

$$f_{12} = \frac{HXY - HXY1}{max\{HX, HY\}} \tag{13}$$

with HXY being the same as f_9. HX and HY are entropies of p_x and p_y respectively, calculated with expression (12). Then

$$HXY1 = -\sum_{i}^{N_g} \sum_{j}^{N_g} p(i,j) \log\{p_x(i) p_y(j)\} \tag{14}$$

It is noteworthy that the contrast and entropy measurements proposed in this subsection are defined over pixel transitions encoded in the GLCM rather than over the raw pixel values.

Measures like homogeneity, contrast or entropy can be quickly interpreted. However, as Haralick *et.al.* stated in [22], not all of these features can be easily understood from a human perspective. This is the case of the Angular Second moment and the IMC. The first can be interpreted as a measure of the energy of the image, but also as a second homogeneity measure. Correlation metrics report the presence of linear dependencies in the image. In the case of the IMC measure, by combining different correlations it provides a reference of the amount of organized structure in the image.

4 Results and Discussion

4.1 Implementation details

Figure 2 shows an overview of the system based on the Zynq UltraScale+ MPSoC ZCU104 architecture. Modern FPGAs can be considered a unit of two main blocks: a Processing System (PS) block and a Programmable Logic (PL) Block. The first normally contains an embedded processor, while the second contains designed hardware [23]. Both blocks are carefully employed in this article, with the PL being used for the majority of the calculations and the PS being used for operations that cannot be parallelized.

Compilation directives can severely affect the systems performance. The `DATAFLOW` directive for instance, plays a critical role in enabling the smooth flow of information between different functions. These, in turn, allow the system to efficiently perform all the necessary operations with a single pass of the image. This is achieved through task-level pipelining, which allows for functions and loops to operate in parallel and overlap, effectively reducing latency and improving the overall throughput of the system. Without the use of the `DATAFLOW` directive, the input image would have to be stored in RAM, thereby reducing the system's efficiency.

The information flow within our proposed system begins with the processing system, which operates on a PetaLinux distribution [24], reading an input image. Subsequently, the image is transformed from RGB to grayscale and the GLCM computed. Next,the image and the computed GLCM are streamed to two distinct

Table 1 Comparison with state-of-the-art research.

Task	Article	Extracted Features	Image Size	Time[ms]	Platform
Histogram Computation	[12]	Histogram	640x360	1.15	Virtex XC4VSX35
	Ours	Histogram	640x480	2.40	ZCU104
	Ours	Histogram	640x480	1.09	AlveoU200
Magnitude image estimation	[13]	Histogram	640x480	98.00	Altera Cyclone IV EP4CE11
	Ours	Histogram	640x480	2.40	ZCU104
	Ours	Histogram	640x480	1.09	AlveoU200
GLCM Computation + Features	[16]	6 Haralick features	512x512	101.50	Virtex-XCV2000
	[16]	6 Haralick features	512x512	37.00	Virtex-XC5VLX50T
	[17]	4 Haralick features	176x144	5.8	ZC702
	Ours	5 Haralick features	640x480	2.40	ZCU104
	Ours	5 Haralick features	640x480	1.09	AlveoU200

Table 2 Latency [ms] for different implementations of our sistem.

	Image Size					
Platform	128x128	320x240	400x300	512x384	640x480	720x480
AlveoU200	0.25	0.31	0.4	0.72	1.09	1.22
ZCU104	0.74	0.85	1.14	1.66	2.40	2.76
PC	104.06	159.11	148.34	230.32	191.42	194.08

blocks: one responsible for generating the histogram and its associated features, and the other for generating the magnitude image and its related metrics. Concurrently, the GLCM-dependent textural features are computed.

Once the parallel computations are completed, results are written into a buffer to return to the PS. Since this research is focused on the optimization and acceleration of the feature extracton process, results are screen printed. Future work may include this information in a control loop.

The source code files were developed in the Xilinx Vitis IDE v2021.2 and Vitis HLS IDE v2021.2. The xfOpenCV library V2022.1 was installed along with the OpenCV V4.4.0 library. GCC 7, G++ 7 and CMake 3.16 were used for compilation.

4.2 Results

Our design was simulated, synthesized, implemented and tested in two separate platforms: a Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit [25] and an Alveo U200 Data Center Accelerator Card [26]. Also, a software version of the system was evaluated for comparison purposes using a computer with an Intel Core i9-10900 CPU running at 2.8 GHz with 16GB of RAM and Ubuntu 20.04 LTS operating system.

Most state-of-the-art research focuses on accelerating only one image operation: performing RGB to gray conversion, histogram calculations, Haralick features extraction, etc. Our proposal, on the other hand, is able to perform multiple computations at the same time with a single pass of the image through the system. Despite this distinction, we provide a comparison of our system to related research in Table 1. Results displayed in this table are those reported by the original authors in their publications.

In terms of histogram computation, our system is compared to [12]. The AlveoU200 implementation of our system and [12] have similar performance, while the ZCU104 needs twice as much processing time.

Our proposal outperforms the work by Tsiktsiris *et. al* [13] in the magnitude image estimation task. The highly parallel nature of the PL part of our system allows for multiple tasks to be performed simultaneously. This condition explains the fact that our measured times for the different tasks are the same.

Siéler *et. al.* [16] proposed a system for Haralick features extraction implemented on two Virtex boards. Table 1 shows that for a 512x512 image the Virtex-XC5VLX50T board performs better than the Virtex-XCV2000 implementation. For a larger image , our proposed system outperforms [16] work in this task, with the exception of estimating one feature less. Finally, the work by Atitallah *et. al.* [17] needs 5.8 ms to calculate the GLCM and four Haralick features for a 176x144 image in a Zc-702 FPGA device.

Table 2 compares the performance of both implementations of the proposed system versus a desktop PC. Latency times were measured feeding the system with different versions of an input image. The performance metric is the time difference between the start of the grayscale image and GLCM stream to the PL and the end of the last metric computation. The aforementioned time measurements were obtained through a combination of C-coded internal calculations on the deployed system and visual inspection of the "Live Waveform Viewer Tool" in Vivado Behavioral Simulation.

While the embedded platforms' calculation times increase with the image size, even for the largest image, they perform in under 3 ms. However, the performance gap between the FPGA implementation and a PC is substantial, around two orders of magnitude.

5 Conclusions

In this paper, low-latency algorithms for for fast and accurate image feature extraction were successfully implemented. A high-speed hardware design provides the user with a set of common statistical image features that can be leveraged for different applications.

Two Xilinx platforms with different hardware resources were used to test the system with various image sizes. Experimental results show that the proposed design is highly efficient, with the ability to complete computations in less than 3 ms, making it appropriate for use in a control loop. These findings strongly suggest that the system is well-suited for integration into autonomous vehicle systems.

Given the use of the Xilinx's xfOpencv library to translate common computer vision operations from sequential software to parallel hardware, it should be easy to extended the proposed system to extract other features or perform different image operations in future developments.

Acknowledgements

J. Gaia would particularly like to thank the High Performance Computing and Networking Research Group of the Escuela Politécnica superior at Universidad Autónoma de Madrid for their cooperation.

Conflict of Interest

No potential conflict of interest was reported by the author(s).

Funding

J.Gaia was funded by the Secretaría de Ciencia, Tecnología e Innovación (SECITI) of San Juan through an "Aportes No Reembolsables(ANR): Formación de Recursos Humanos - Estadías Científicas" program by resolution number 022-SECITI-2020.

References

1. Y. Kadokawa, Y. Tsurumine, and T. Matsubara, "Binarized p-network: Deep reinforcement learning of robot control from raw images on fpga," *IEEE Robotics and Automation Letters*, 2021.
2. M. Ravi, A. Sewa, T. Shashidhar, and S. S. S. Sanagapati, "Fpga as a hardware accelerator for computation intensive maximum likelihood expectation maximization medical image reconstruction algorithm," *IEEE Access*, 2019.
3. X. Jiang, "Human tracking of track and field athletes based on fpga and computer vision," *Microprocessors and Microsystems*, 2021.
4. S. Aldegheri, N. Bombieri, D. D. Bloisi, and A. Farinelli, "Data flow orb-slam for real-time performance on embedded gpu boards," in *2019 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2019.
5. S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proceedings of the IEEE*, 2019.
6. J. Webber, A. Mehbodniya, R. Teng, A. Arafa, and A. Alwakeel, "Finger-gesture recognition for visible light communication systems using machine learning," *Applied Sciences*, 2021.
7. S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with gpus and fpgas," in *2008 Symposium on Application Specific Processors*. IEEE, 2008.
8. D. Honegger, H. Oleynikova, and M. Pollefeys, "Real-time and low latency embedded computer vision hardware based on a combination of fpga and mobile cpu," in *2014 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. IEEE, 2014.
9. M. Bouain, K. M. Ali, D. Berdjag, N. Fakhfakh, and R. B. Atitallah, "An embedded multi-sensor data fusion design for vehicle perception tasks." *J. Commun.*, vol. 13, no. 1, pp. 8–14, 2018.
10. G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.
11. Y. Zhang, X. Yang, L. Wu, and J. H. Andrian, "A case study on approximate fpga design with an open-source image processing platform," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2019.
12. D. Younis and B. M. Younis, "Low cost histogram implementation for image processing using fpga," in *IOP Conference Series: Materials Science and Engineering*. IOP Publishing, 2020.
13. D. Tsiktsiris, D. Ziouzios, and M. Dasygenis, "A portable image processing accelerator using fpga," in *2018 7th Int. Conf. on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, 2018.
14. A. Linares-Barranco, F. Perez-Pena, D. P. Moeys, F. Gomez-Rodriguez, G. Jimenez-Moreno, S.-C. Liu, and T. Delbruck, "Low latency event-based filtering and feature extraction for dynamic vision sensors in real-time fpga applications," *IEEE Access*, 2019.
15. F. Siddiqui, S. Amiri, U. I. Minhas, T. Deng, R. Woods, K. Rafferty, and D. Crookes, "Fpga-based processor acceleration for image processing applications," *Journal of Imaging*, 2019.
16. L. Siéler, C. Tanougast, and A. Bouridane, "A scalable and embedded fpga architecture for efficient computation of grey level co-occurrence matrices and haralick textures features," *Microprocessors and Microsystems*, 2010.
17. M. A. B. Atitallah, R. Kachouri, and H. Mnif, "A new fpga accelerator based on circular buffer unit per orientation for a fast and optimised glcm and texture feature computation," in *2019 IEEE International Conference on Design & Test of Integrated Micro & Nano-Systems (DTS)*. IEEE, 2019, pp. 1–6.
18. F. Vahid, *Digital design with RTL design, VHDL, and Verilog*. John Wiley & Sons, 2010.
19. Xilinx, "Xilinx opencv library," https://github.com/Xilinx/xfopencv, 2017, [Online; accessed 13-February-2023].

20. A. K. Tripathi, S. Mukhopadhyay, and A. K. Dhara, “Performance metrics for image contrast,” in *2011 Int. Conf. on Image Information Processing.* IEEE, 2011.
21. U. Shin, J. Park, G. Shim, F. Rameau, and I. S. Kweon, “Camera exposure control for robust robot vision with noise-aware image quality assessment,” in *2019 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS).* IEEE, 2019.
22. R. M. Haralick, K. Shanmugam, and I. H. Dinstein, “Textural features for image classification,” *IEEE Transactions on systems, man, and cybernetics*, 1973.
23. Y. Nitta, S. Tamura, H. Yugen, and H. Takase, “Zytlebot: Fpga integrated development platform for ros based autonomous mobile robot,” in *2019 Int. Conf. on Field-Programmable Technology (ICFPT).* IEEE, 2019.
24. Xilinx, “Petalinux tools,” https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html, 2022, [Online; accessed 16-February-2023].
25. ——, “Zynq ultrascale+ mpsoc zcu104 evaluation kit,” https://www.xilinx.com/products/boards-and-kits/zcu104.html, 2022, [Online; accessed 21-February-2023].
26. ——, “Zynq ultrascale+ mpsoc zcu104 evaluation kit,” https://www.xilinx.com/products/boards-and-kits/alveo/u200.html, 2022, [Online; accessed 21-February-2023].

License