applied sciences

MDPI

Article

TurboPixels: A Superpixel Segmentation Algorithm Suitable for Real-Time Embedded Applications

Abiel Aguilar-González [1,*], Alejandro Medina Santiago [1,*], Jorge Antonio Orozco Torres [2], J. A. de Jesús Osuna-Coutiño [3], Madaín Pérez Patricio [3] and Néstor A. Morales-Navarro [3]

[1] Computer Science Department, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), San Andrés Cholula 72840, Mexico
[2] Industrial Engineering Department, Tecnológico Nacional de México, Instituto Tecnológico de Tuxtla Gutiérrez (ITTG), Tuxtla Gutiérrez 29050, Mexico
[3] Mechatronics Engineering Department, Tecnológico Nacional de México, Instituto Tecnológico de Tuxtla Gutiérrez (ITTG), Tuxtla Gutiérrez 29050, Mexico
* Correspondence: abiel@inaoep.mx (A.A.-G.); amedina@inaoep.mx (A.M.S.)

Abstract: Superpixel segmentation aims to produce a consistent grouping of pixels. In recent years, the importance of superpixel segmentation has increased in computer vision since it offers useful primitives for extracting image features and simplifies the complexity of other image processing steps. In this work, we propose the TurboPixels algorithm, whose main contribution is a hardware architecture for superpixel segmentation. Compared with previous approaches, our superpixels are computed without the need for iterative loops. This makes it possible to reduce algorithmic complexity and increases processing speed. The experimental results indicate that our approach enables a small-scale FPGA-based implementation suitable for embedded applications. In addition, the results demonstrate that robust superpixel segmentation can be achieved with processing speeds up to 86 times faster than in previous works in the current literature.

Keywords: image processing; image segmentation; real-time processing; FPGA; VHDL

check for updates

Citation: Aguilar-González, A.; Medina Santiago, A.; Orozco Torres, J.A.; Osuna-Coutiño, J.A.d.J.; Pérez Patricio, M.; Morales-Navarro, N.A. TurboPixels: A Superpixel Segmentation Algorithm Suitable for Real-Time Embedded Applications. *Appl. Sci.* **2024**, *14*, 11912. https://doi.org/10.3390/app142411912

Academic Editor: Pedro Alexandre Mogadouro do Couto

Received: 18 October 2024
Revised: 8 December 2024
Accepted: 16 December 2024
Published: 19 December 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In recent years, superpixels have increased their popularity in computer vision applications. However, the previous works regarding superpixel segmentation suffer from high computational costs, which makes them unable to be used in real-time systems. To address this problem, in most cases, Simple Linear Iterative Clustering (SLIC) is used [1]. There are multiple SLIC-CUDA implementations that aim to address the real-time processing scenario [2,3]. The SLIC algorithm was developed to efficiently generate compact and nearly uniform superpixels. Its simplicity, efficiency, and performance make it a practical option for real-time systems compared to other superpixel approaches in the current state of the art, such as Normalized Cuts [4] and QuickShift [5]. However, the CPU-sequential implementation of SLIC takes approximately 300–400 ms to segment a 640 × 480 image. Although decreasing the number of iterations for each cluster can speed up the algorithm, this approach may result in a loss of performance.

In addition, several superpixel segmentation algorithms have been proposed in the recent literature [6–9]. The significance of superpixels lies in their applications in multiple domains, including object recognition [10], image segmentation [11], object tracking [12], video segmentation [13,14], classification [15], and reconstruction [16]. Each approach offers advantages and disadvantages that demonstrate the importance of the properties of these algorithms. In the existing literature, the following properties are key for robust superpixel segmentation:

1. Accuracy: Measures how well superpixels align with image boundaries. To retain more information for subsequent processing on superpixels, segmentation should strive for maximum accuracy.
2. Regularity: Evaluates how closely a superpixel approximates a square shape. Superpixels are utilized to represent pixel information, and regular superpixels can maintain typical pixel characteristics, including spatial topology, structure, homogeneity, and isometric information.
3. Efficiency: Serving as a preprocessing step in image processing/computer vision, superpixels aim to reduce computational complexity. They should be fast, memory-efficient, and easy to utilize.

Superpixel Segmentation

Superpixel segmentation aims to divide an image into multiple segments (collections of pixels or superpixels), as illustrated in Figure 1. The objective is to transform the image into a representation that is more meaningful and easier to analyze. Image segmentation aims to assign a label to each pixel within an image such that pixels sharing the same label exhibit certain similarities. This technique can be used to identify objects and delineate boundaries (such as lines and curves) within images [17]. The technical advantages of superpixels include the following:

1. Computational Efficiency: Although computing the actual groupings of superpixels can be somewhat resource-intensive, it significantly reduces the overall complexity of images, from hundreds of thousands of pixels to just a few hundred superpixels. Each superpixel encapsulates some form of perceptual, ideally semantic, information.
2. Perceptual Meaningfulness: Instead of focusing on individual pixels within a grid, which possess minimal perceptual significance, superpixels group together pixels that share common characteristics, such as similar color or texture patterns.
3. Oversegmentation: Many superpixel algorithms aim to oversegment images. This results in the identification of the most important boundaries within the image. Although this may initially appear to be problematic or a drawback of using superpixels, it is beneficial because, thanks to this oversegmentation, very few (if any) pixels are lost when transitioning from pixel grids to superpixel mappings.

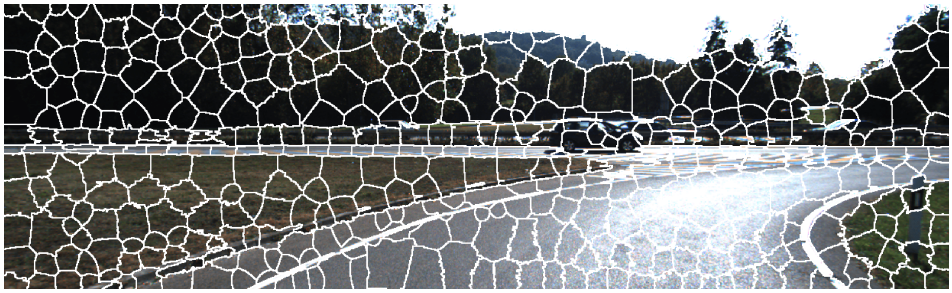


Figure 1. Superpixel segmentation in which each region represents a superpixel.

2. Related Works

The superpixel segmentation problem has attracted significant attention in the field of computer vision due to its ability to reduce image complexity while preserving essential structural information. Over the years, multiple algorithms have been developed to generate superpixels efficiently. The early approaches, such as the Normalized Cuts (NCut) algorithm, focused on partitioning images based on global optimization criteria by formulating the segmentation problem as a graph partitioning task [4]. Although this method provided accurate segmentation, its reliance on solving eigenvector problems resulted in high computational costs, making it unsuitable for real-time applications. Subsequent methods were designed to address these limitations. Quickshift [5], for example, is a mode-seeking algorithm that clusters pixels based on kernel density estimation in

feature space, offering a balance between segmentation quality and computational cost, but its efficiency decreases significantly for high-resolution images. Felzenszwalb's efficient graph-based segmentation [18] introduced a hierarchical approach that considers both global and local image properties, making it computationally efficient and suitable for a variety of applications. The SLIC algorithm [1] further advanced the field by employing a k-means-based approach tailored specifically for superpixel generation. SLIC optimizes computational efficiency by limiting the clustering to a local search region, enabling fast and scalable performance. Among these, SLIC has become one of the most widely used techniques due to its simplicity, speed, and ability to produce superpixels with uniform sizes and shapes, making it a standard baseline for comparison in many studies.

Despite the success of these software-based algorithms, the growing demand for real-time applications, such as autonomous driving, robotics, and medical imaging, has driven research toward hardware-accelerated solutions. Several studies have explored hardware-friendly adaptations of traditional algorithms to take advantage of the parallelism offered by Field-Programmable Gate Arrays (FPGAs) and Graphics Processing Units (GPUs). These efforts aim to bridge the gap between segmentation quality and processing speed, enabling real-time performance for high-resolution images. However, many of these approaches involve compromises in algorithmic complexity or require significant hardware resources, motivating further investigation into tailored algorithms and architectures for hardware implementations. In Ref. [19], an optimized hardware pipelined implementation of the accelerated Simple Linear Iterative Clustering (A-SLIC) algorithm for superpixel segmentation was introduced. This work proposed a hardware–software codesign framework that utilizes off-chip memory to manage large memory demands. Compared to the traditional sequential software implementation of SLIC running on a CPU, the algorithm executed on the Zynq 7000 device (Xilinx, Inc., San Jose, CA, USA) demonstrates a significant speed increase, achieving a performance improvement of 10 to 22 times.

In Ref. [20], a fully automated complete FPGA architecture was introduced utilizing SLIC, referred to as FP-SLIC. This design offers several key features: (1) a streamlined and efficient algorithm that significantly decreases computational complexity, facilitating the development of algorithms for FPGAs; (2) a fully pipelined FPGA configuration that operates at 40 MHz with a throughput of one pixel per cycle; and (3) a memory-efficient architecture that removes the need for external memory. The proposed algorithm achieved remarkable results on the BSDS500 benchmark (University of California, Berkeley, CA, USA), particularly in terms of boundary recall for fewer than 1000 superpixels, outperforming related approaches while maintaining a processing speed of 259 frames per second (fps).

In Ref. [19], a GPU implementation of the Quick Shift image segmentation algorithm was presented, showcasing significant advancements in computational efficiency for superpixel generation. Although this approach achieves impressive speedups, its reliance on texture caching and relatively limited throughput at larger resolutions may pose challenges for high-definition applications. Compared to FPGA-based designs, such as FP-SLIC [20], which achieve throughput rates of up to 259 frames per second on high-resolution benchmarks, the GPU implementation trades off higher scalability and resolution capabilities for simpler implementation and cost-effective hardware requirements.

In this work, we propose a new algorithm and a hardware architecture suitable for embedded implementation. Unlike previous works, which adapt well-known superpixel segmentation algorithms to fit the implementation constraints of FPGAs, our algorithm was initially formulated to exploit the parallelism and pipeline flow of a dedicated hardware architecture. This enables us to achieve robust superpixel segmentation with a speedup of up to $\times 16$ and lower hardware resource usage compared to previous approaches.

3. The Proposed Algorithm

In Figure 2, an overview of our algorithm is shown. Our approach consists of an image preprocessing step in which bits per pixel (bpp) of the input image are reduced. Then, in a

second step, a superpixel mesh (M) and median "seed" value (m) are created. Finally, the bpp-reduced image (I'), median value (m), and mesh template (M) are used to define the superpixels.

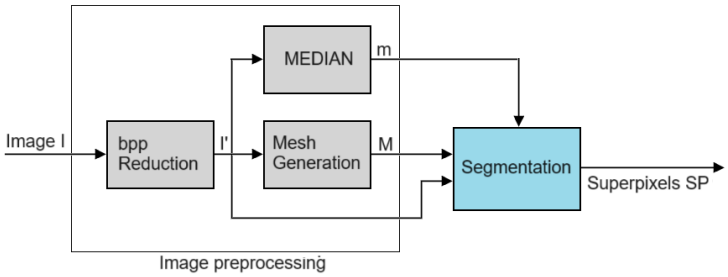


Figure 2. Block diagram of the proposed algorithm.

3.1. bpp Reduction

Given a grayscale image $I(i,j)$, as presented in Figure 3, a bpp reduction over the image I is carried out to reduce noise and remove fine-scale structures that affect the performance of the segmentation step. The reduction in bpp is defined by Equation (1) as follows:

$$I'(i,j) = I(i,j)/2^{\alpha} \tag{1}$$

where α is the reduction factor whose value is defined as int $0 \leq \alpha \leq$ int Mbpp; Mbpp is the color depth (max bits per pixel). In Figure 4, an example of bpp reduction is shown. The main objective of this reduction is to minimize noise and remove fine-scale details that could interfere with the superpixel segmentation process. Before the bpp reduction (Figure 3), the input image contains a high level of pixel-level information, which includes finer details, such as noise and subtle texture variations. These elements may not be crucial for superpixel segmentation, which aims to group similar pixels together. Therefore, retaining these fine details may result in unnecessary complexity, making the segmentation process more computationally intensive.

After the bpp reduction, as shown in Figure 4, the image appears to be smoother, with fewer subtle variations in pixel intensity. The reduction helps to suppress noise and eliminate some of the finer details that do not contribute significantly to the segmentation process. This step effectively reduces the image's complexity, enabling more efficient processing while still preserving the key structural features necessary for accurate segmentation. Additionally, bpp reduction helps to enhance the robustness of superpixel segmentation, particularly in scenarios where high-resolution or noisy images are involved. By eliminating non-essential information, the algorithm focuses on the broader image structure, which is more important for identifying the object boundaries and generating meaningful superpixels.



Figure 3. Input image.



Figure 4. bpp-reduced image.

3.2. Mesh Generation

The second step consists of the creation of a rough superpixel mesh (M); let $(2 \times s) + 1$ be the superpixel "seed" size; the superpixel mesh is defined as follows:

$$M(i,j) = M' \tag{2}$$

$$M'(k-s:k+s, v-s:v+s) = L_{k,v} \tag{3}$$

where $k = \lfloor x_{size}/(2 \times s) + 1 \rfloor$, $v = \lfloor y_{size}/(2 \times s) + 1 \rfloor$, $L_{k,v} = k + v \times \lfloor (y/(2 \times s) + 1) \rfloor$, (x, y) is the input image resolution, and s defines the initial size of the "seed" superpixel, as mentioned before. In Figure 5, an example of a superpixel mesh is shown. For each seed (in Figure 5), examples of three different seeds are presented; a median value $(m(i,j))$ is computed as follows:

$$m(i,j) = m' \tag{4}$$

$$m'(k-s:k+s, v-s:v+s) = \text{MEAN}(I'(k-s:k+s, v-s:v+s)) \tag{5}$$

where $k = \lfloor x_{size}/(2 \times s) + 1 \rfloor$, $v = \lfloor y_{size}/(2 \times s) + 1 \rfloor$, (x, y) is the input image resolution and I' is the bpp reduction (Equation (1)).

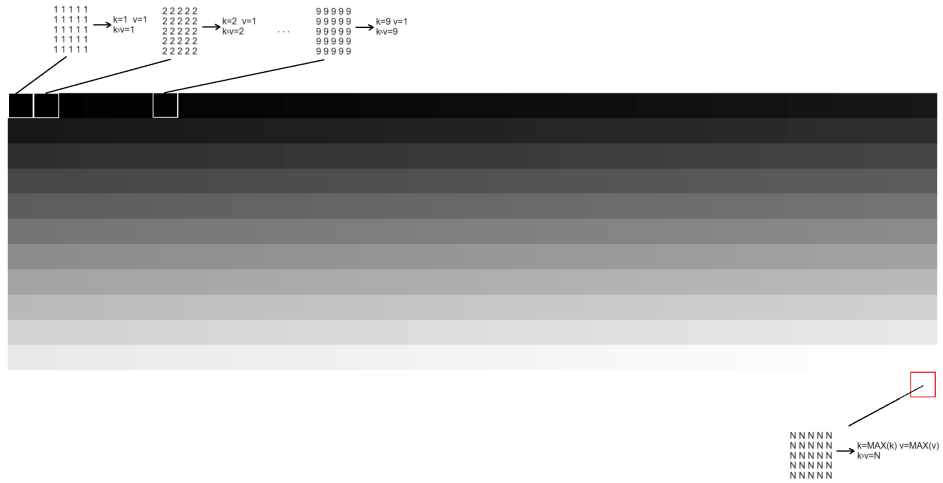


Figure 5. Superpixel mesh (Equation (2)). The initial "seeds" (white bounding box) are defined as a grid of symmetric squares of size equal to $(2 \times s) + 1$.

The primary motivation for using a superpixel mesh is to simplify and accelerate the image segmentation process. In traditional image segmentation, pixel-wise classification can be computationally expensive and ineffective, especially in high-resolution images. Superpixels, which group pixels with similar characteristics into regions, enable a more compact and meaningful representation of an image. By using a superpixel mesh, we can ensure that these regions are structured in a regular grid, which not only reduces computational complexity but also maintains essential image information while facilitating faster processing.

The mesh serves as a structured framework for defining and locating superpixels within the image, ensuring that the segmentation is conducted uniformly and efficiently. This approach improves the performance of the algorithm by reducing the number of operations required to process each pixel individually, thus significantly speeding up the segmentation process. In addition, the regularity of the superpixel mesh ensures that the resulting segments are consistent, improving the quality of image analysis tasks such as feature extraction, object recognition, and boundary detection.

Segmentation

The segmentation step refines the superpixel seeds generated in the previous stages, constraining and growing them to form the final superpixel structure. Initially, each "seed" corresponds to a potential superpixel. Using the mean values computed in Equation (4), the seeds are iteratively adjusted so that all pixels within a superpixel share a similar gray value. This process ensures high regularity and precision in segmentation.

The segmentation process is mathematically defined as follows:

$$SP(i,j) = M(i,j) \quad \text{where} \quad (u,v) = \arg\min_{u,v} \left| I'(i,j) - m(i+u, j+v) \right| \tag{6}$$

Here, the final superpixel label, $SP(i,j)$, is assigned to each pixel (i,j) by associating it with the nearest "seed" median value $m(i+u, j+v)$ that minimizes the difference in gray values. Five candidate neighbors are considered, defined by $u = 0, -s-1, 0, s+1, 0$ and $v = -s-1, 0, 0, 0, s+1$. This ensures that the segmentation captures the local intensity uniformity around each seed.

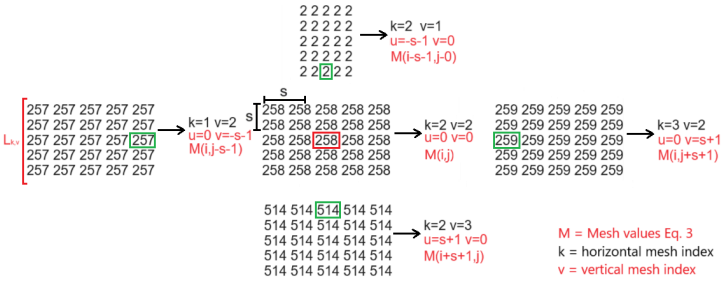
To further clarify the process, the following pseudocode outlines the Algorithm 1:

Algorithm 1: Superpixel Segmentation

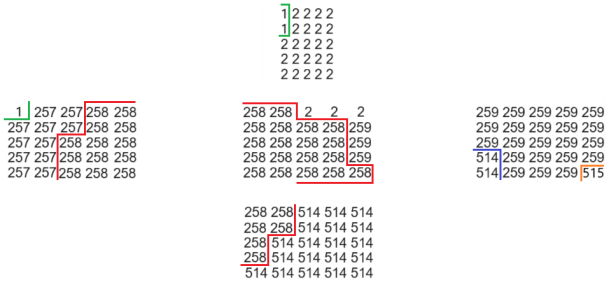
Input: Reduced image I', Superpixel mesh M, Median values m
Output: Final superpixel labels SP
for each pixel (i,j) in I' **do**
  Initialize minimum_diff $\leftarrow \infty$
  for each neighboring seed (u,v) **do**
    Compute diff $\leftarrow |I'(i,j) - m(i+u, j+v)|$
    if diff $<$ minimum_diff **then**
      Update minimum_diff $\leftarrow$ diff
      Assign $SP(i,j) \leftarrow M(i+u, j+v)$
    end if
  end for
end for
return SP

Figure 6a demonstrates the iterative assignment process, where each pixel is compared to neighboring seeds. Figure 6b highlights the outcome of this segmentation, showing how seeds grow to form cohesive superpixels. For example, the center seed labeled "248" expands downward and to the left, forming superpixel "258". The high regularity and uniformity achieved demonstrate the effectiveness of the proposed approach.

This process not only reduces the complexity of subsequent image analysis tasks but also provides a robust preprocessing step for applications requiring precise region segmentation.



(**a**) The segmentation process



(**b**) Outcome of the segmentation process

Figure 6. Segmentation process. (**a**) For each pixel (red bounding box) in the input image; bpp-reduced image. $I'(i,j)$ are compared with median values of the neighbor "seeds" (green bounding boxes) in order to add or remove pixels in the "seeds" previously computed (Equation (2)). All pixels in the "seed" share median values. (**b**) Outcome of the segmentation process: the center seed labeled "248" has grown downwards and to the left, forming the superpixel labeled "258", as shown in the image. Additionally, other seeds, such as "1" and "515", are now visible in the mesh section.

4. FPGA Architecture

An overview of the FPGA architecture for the proposed algorithm is presented in Figure 7. This architecture focuses on an FPGA implementation in which all recursive and parallelizable operations are accelerated inside the FPGA fabric. Initially, the "bpp Reduction" unit processes the pixel stream (I [7:0]) provided by the imager, executing the bpp reduction process, as defined in Equation (1). The "Data Preprocessing" unit implements "circular buffers" in order to temporarily store local sections of the frame being processed, enabling local parallel access that supports concurrent processing for computing the "superpixel mesh" and the associated median values (Equations (2) and (4)). Finally, the streams for "mesh" and "median values" (m [3:0], M [15:0]) are used to compute the superpixel segmentation (SP [15:0]). In the subsequent subsections, we provide further details regarding the parallelization of the algorithm.

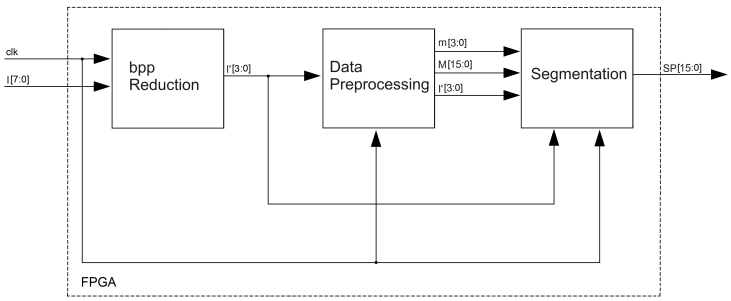


Figure 7. FPGA architecture for the proposed algorithm.

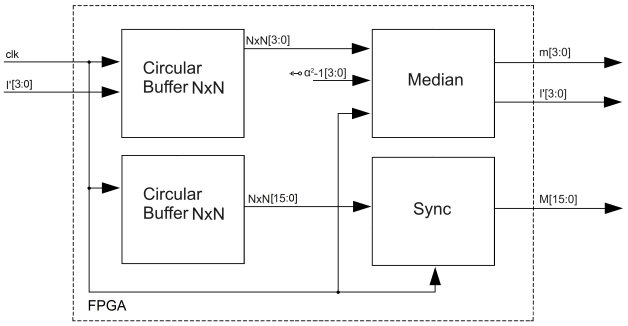
4.1. bpp Reduction

Regarding hardware implementation, Equation (1) can be implemented directly using shift operators as follows: $I'(i,j) = \texttt{srl}_{\alpha}(I(i,j))$, where $\texttt{srl}_{\alpha}$ corresponds to a right shift

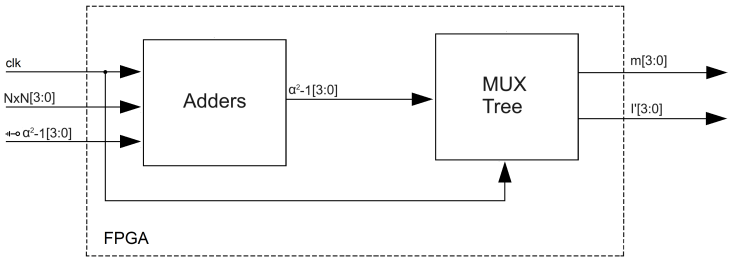
over a register and the magnitude of the shift is determined by α. This will require the minimum hardware resource usage and will enable real-time processing.

4.2. Data Preprocessing

In Figure 8, the FPGA architecture for the "Data Preprocessing" unit is shown. The behavior of the buffers is modified from the original formulation (Figure 9) in order to repeat values up to $(2 \times s) + 1 \times (2 \times s) + 1$ and therefore be capable of generating the "superpixel" mesh (Equation (2)). The required behavior can be achieved by modification of the "control" unit and adding $2 \times n + 1$ BRAM modules instead of $n + 1$. In this case, the buffer size N is defined as $(2 \times \theta)$, where $\theta = (2 \times s) + 1$ and s is the "seed" size for the superpixel mesh. These buffers enable us to maintain all the data as a stream. Then, median values can be computed by adding votes at each possible value depending on the reduction factor (α; see Equation (1)). Finally, a multiplexer tree (MUX) that determines the value with the highest votes defines the median value, as shown in Figure 8b.



(**a**) FPGA architecture for the "Data Preprocessing" unit



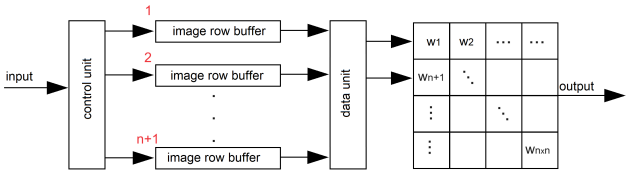
(**b**) FPGA architecture for the "Median" module

Figure 8. FPGA architecture for the proposed algorithm. (**a**) The architecture for the 'Data Preprocessing' unit includes modified buffers and a control unit for generating the superpixel mesh. (**b**) The architecture for the 'Median' module, which determines median values using a multiplexer tree based on a voting mechanism.

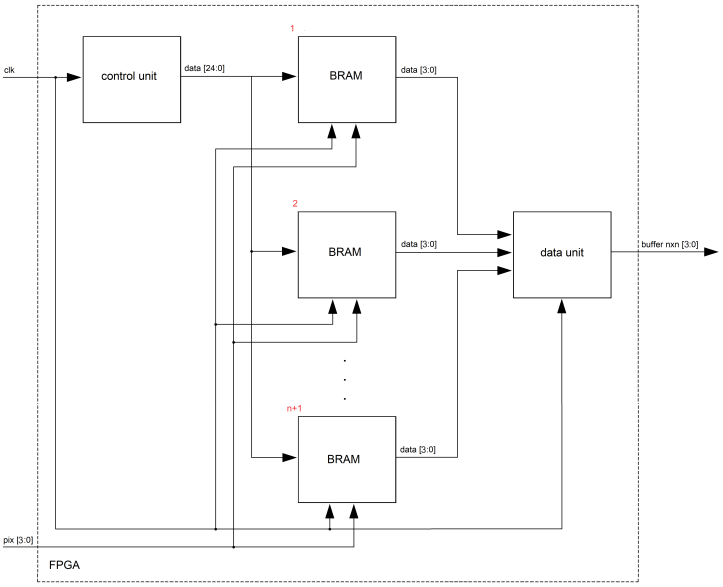
Circular Buffer

In Ref. [21], a circular buffer architecture was proposed. This architecture enables the storage of input data from the previous n rows of an image using memory buffers (block RAMs/BRAMs) until a $n \times n$ neighborhood is scanned along subsequent rows. In this work, we adopt a similar implementation in order to achieve enhanced data reuse and a high level of parallelism. Our algorithm is processed in modules, enabling the parallel reading of all image patches. Initially, a "control" unit for the shift mechanism governs the read/write addresses for $n + 1$ BRAMs; in this configuration, n BRAMs operate in read mode while one BRAM is in write mode during each clock cycle. Then, data in read mode BRAMs can be accessed concurrently, enabling each pixel within an $n \times n$ area to be output in parallel to an $n \times n$ buffer, as illustrated in Figure 9. The "control" unit sends control signals (address and read/write enabled) to the BRAM modules, with one complete row stored in each BRAM. Finally, the "data" unit delivers $n \times n$ pixels simultaneously. In our

implementation, there are two circular buffers containing $n \times n$ pixels/bytes, where the size n corresponds to the initial size of the "seed" $((2 \times s) + 1)$ described in the previous section. The first buffer holds the $M(i, j)$ "mesh" values, requiring a [15:0] byte depth, while the second buffer manages the $m(i, j)$ "median" values, thus using a [3:0] byte depth, as illustrated in Figure 9.



(**a**) General formulation of an $n \times n$ circular buffer

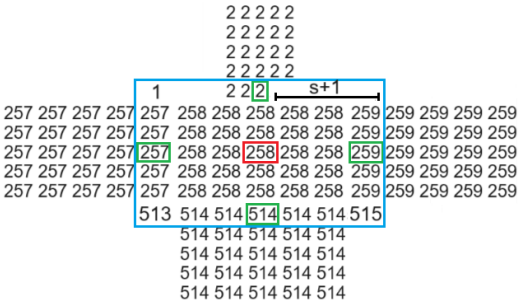


(**b**) FPGA architecture for the circular buffers

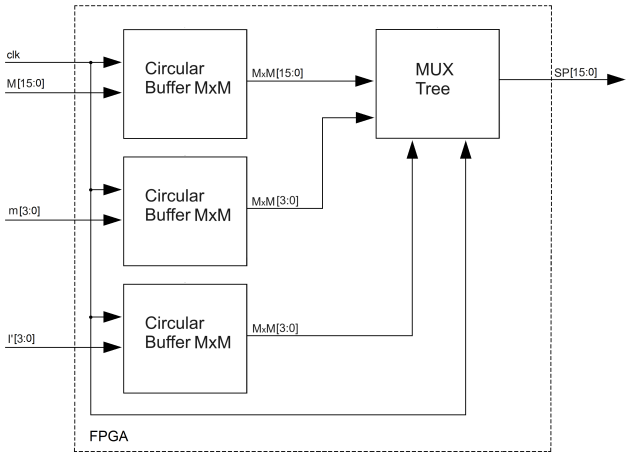
Figure 9. The circular buffers' architecture. For an $n \times n$ patch, a shift mechanism "control" unit manages the read/write addresses of $n + 1$ BRAMs. In this formulation, n BRAMs are in read mode and one BRAM is in write mode in each clock cycle. Then, the $n \times n$ buffer delivers logic registers with all pixels within the patch in parallel.

4.3. Segmentation

In Figure 10, the FPGA architecture for the "Segmentation" unit is shown. The first $M \times M$ buffers for the "Data Preprocessing" output are implemented. M is defined as $(2 \times \beta) + 1$, where $\beta = s + 1$ and s are the "seed" sizes for the superpixel mesh. These buffers enable processing as stream, as shown in Figure 8a, the comparisons between the gray value $(I'(i, j))$ of the pixel being processed (red bounding box) and the neighbors' median values $(m(i, j))$, green bounding box. The label $(M(i, j))$ of the candidate that minimizes the function (Equation (6)) is set as result, and, therefore, the pixels being processed now are bundled inside superpixel $L_{k,v}$; see Equation (3). This process is carried out through a classical implementation of multiplexor tree (MUX); for more details, please see [22].



(**a**) Segmentation buffer distribution



(**b**) FPGA architecture for the "Segmentation" unit

Figure 10. $M \times M$ buffers enable processing as stream. Comparisons between the gray value $(I'(i,j))$ of the pixel being processed (red bounding box) and the neighbors' median values $(m(i,j))$; green bounding boxes are computed. The label of the candidate $(M(i,j))$, which minimizes the function (Equation (6)), bundles the pixel being processed inside the superpixel $L_{k,v}$.

5. Results

The developed FPGA architecture was implemented on an Altera FPGA Cyclone IV EP4CGX150CF23C8. All the modules were designed via Quartus II Web Edition version 10.1SP1, and all the modules were validated via post-synthesis simulations performed in ModelSim Altera. For all the tests, we consider $\alpha = 4$, $s = 8$ (Equations (1) and (3)) since these values provided relatively good performance for real-world scenarios. In practice, we recommend these values as a reference. The detailed consumption of hardware resources for the architecture is shown in Table 1. Our algorithm formulation enables a compact system design; it requires 41% of the total logic elements of the FPGA Cyclone IV EP4CGX150CF23C8. Regarding memory resources, our architecture requires 57% of the total available resources, which translates into the consumption of 119 block RAMs, primarily for circular buffers. This level of hardware utilization enables targeting a relatively compact FPGA device, which makes it well suited for real-time embedded applications. The subsequent subsections will present comparisons with previous work. We will provide a discussion of the performance and limitations of the proposed algorithm in relation to the current state of the art.

Table 1. Hardware resource consumption for the developed FPGA architecture.

Resource	Consumption/"Seed" Size		
	$s = 8$	$s = 4$	$s = 2$
Total logic elements	41,182 (41%)	31,978 (27%)	27,279 (23%)
Total pins	24 (5%)	24 (5%)	24 (3%)
Total Memory Bits	1,722,355 (57%)	926,382 (24%)	619,145 (16%)
Embedded multiplier elements	0 (0%)	0 (0%)	0 (0%)
Total PLLs	0 (0%)	0 (0%)	1 (0%)

Operating frequency = 50 MHz.

In Table 2, we provide a comparative analysis of the utilization of hardware resources between our FPGA architecture and previous FPGA-based superpixel segmentation implementations. As mentioned above, the existing state-of-the-art hardware implementations are somewhat constrained. Nevertheless, in recent years, several works have emerged that focus on enhancing performance in terms of processing speed and computational efficiency via dedicated hardware architectures. Compared with earlier studies, our FPGA architecture demonstrates superior performance, requiring fewer logic elements and memory resources while achieving comparable operational frequencies (MHz). Specifically, it outperforms the implementation described in [19] in both logic elements and memory utilization, as well as indicating a reduced logic element count compared to [20]. The advancements in our architecture not only reflect improvements in resource efficiency but also highlight its potential applicability in real-time embedded systems. The reduced hardware footprint enables more versatile deployments in several applications, making our design a significant step forward in the realm of FPGA-based superpixel segmentation.

Table 2. Hardware resource consumption comparisons.

Method	Logic Elements	Memory Bits	MHz
Ghaderi et al. [20] (2022)	$3 \times 19{,}964$	3×24 BRAMs	40
Ghimire et al. [19] (2020)	52,382	266 BRAMs	82
Our work ($S = 8$) *	41,182	119 BRAMs	50

* Operating frequency = 50 MHz.

In Table 3, we present the processing speeds achieved for multiple image resolutions. We developed and synthesized multiple versions of our FPGA architecture, as illustrated in Figure 7. Following the synthesis process, we conducted post-synthesis simulations using ModelSim Altera. In each scenario, our FPGA architecture demonstrated the capability of real-time processing. Compared to the previous work, as shown in Table 4, our algorithm delivered the highest processing speeds, outperforming several previous implementations [19,20]. In particular, for full-HD images, our algorithm achieves real-time processing rates higher than 30 frames per second (fps) for a resolution of 1920×1080. These results demonstrate the efficiency and effectiveness of our proposed architecture for high-resolution images, reinforcing its potential for use in demanding real-time applications such as autonomous vehicles and video analytics. The ability to maintain high processing speeds while utilizing fewer hardware resources positions our approach as a robust solution in the domain of FPGA-based image processing.

Table 3. Processing speeds for different image resolutions.

Resolution	Frames/s	Pixels/s
1920×1080	37	76,723,200
1280×720	81	80,179,200
640×480	297	91,238,400
320×240	1209	92,880,000

Operating frequency = 50 MHz.

Table 4. Processing speed comparisons.

Method	Resolution	Frames/s	Pixels/s
Ghaderi et al. [20] (2022)	320 × 240	527	40,473,600
Ghimire et al. [19] (2020)	321 × 321	14	1,442,574
Our work	320 × 240	1209	92,880,000

Finally, qualitative results for our superpixel segmentation algorithm are presented in Figures 11 and 12. Although we prefer to present quantitative results, it is important to note that the prior works did not typically share their datasets, opting instead to use their own images. In our work, we used random images sourced from the KITTI dataset [23]. In all cases, our algorithm successfully achieves precise superpixel segmentation. These qualitative results demonstrate not only the robustness of our algorithm but also its capability to generalize across diverse image types. The visual representations demonstrated that our approach effectively delineates object boundaries and preserves significant image details, reinforcing its potential applicability in multiple computer vision tasks such as object detection and tracking. Furthermore, the use of the KITTI dataset enables a standardized comparison, which enhances the reliability of our findings in the context of the current segmentation techniques.

Based on the metrics presented in Section 1, Figure 11 demonstrates the advantages of our proposed method, showing its superior regularity while maintaining a high level of accuracy. This balance sets our method apart from the previous approaches, particularly in terms of efficiency. The dedicated hardware implementation ensures the fastest processing speeds among comparable methods, with the resource consumption optimized for embedded applications.

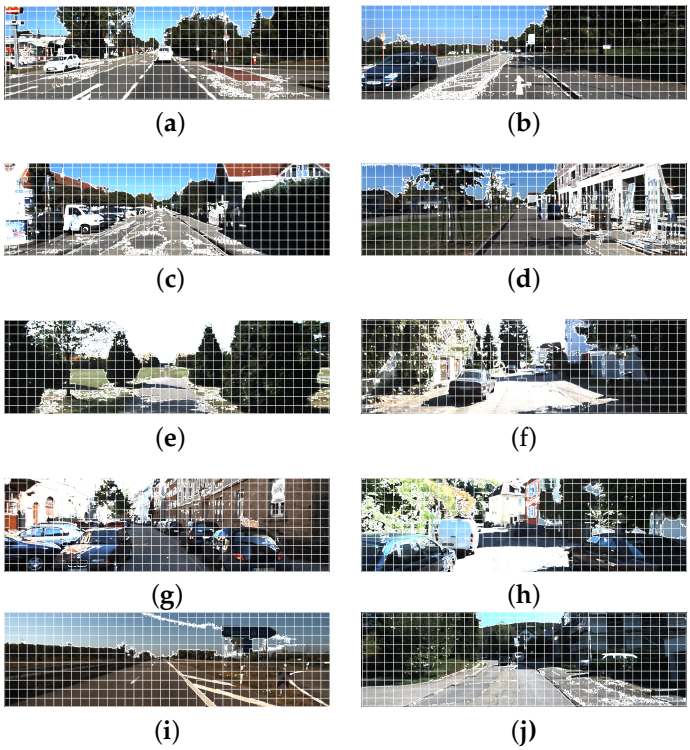


Figure 11. Superpixel Segmentation: Quantitative results for the KITTI dataset ($s = 8$). The subfigures (**a**–**j**) depict images from different sequences of the KITTI dataset in different urban environment contexts. Therefore, this results demonstrate the advantages of the proposed method, highlighting its high regularity and accuracy. The uniformity in superpixel sizes and shapes illustrates the exceptional regularity achieved, which outperforms prior works.

What further distinguishes our method is the exceptional level of regularity achieved, which, to our knowledge, surpasses that of any prior work. This high regularity is critical for ensuring uniform superpixel sizes and shapes, which contributes to more consistent segmentation outcomes. This consistency is especially valuable for applications that require precise boundary delineation or uniform input features, such as object recognition, scene segmentation, or medical image analysis. By achieving this high level of regularity, our method reduces irregularities that might otherwise lead to errors or inconsistencies in the subsequent image processing steps. This makes our approach not only more accurate and efficient but also more robust across diverse application scenarios. Thus, the proposed method represents a significant advancement in superpixel segmentation, offering unmatched regularity alongside optimal performance metrics.

In contrast, Figure 12 illustrates how increasing the number of superpixels enhances regularity while maintaining accuracy. However, this increase comes with potential drawbacks when considering efficiency for image processing or analysis tasks. A higher number of superpixels implies a greater computational requirement, potentially reducing the overall efficiency. In comparison, the results presented in Figure 11 offer a more balanced trade-off between precision and regularity while significantly reducing the number of superpixels. This reduction translates into fewer potential operations required for subsequent image analysis algorithms, ensuring a more efficient processing pipeline.

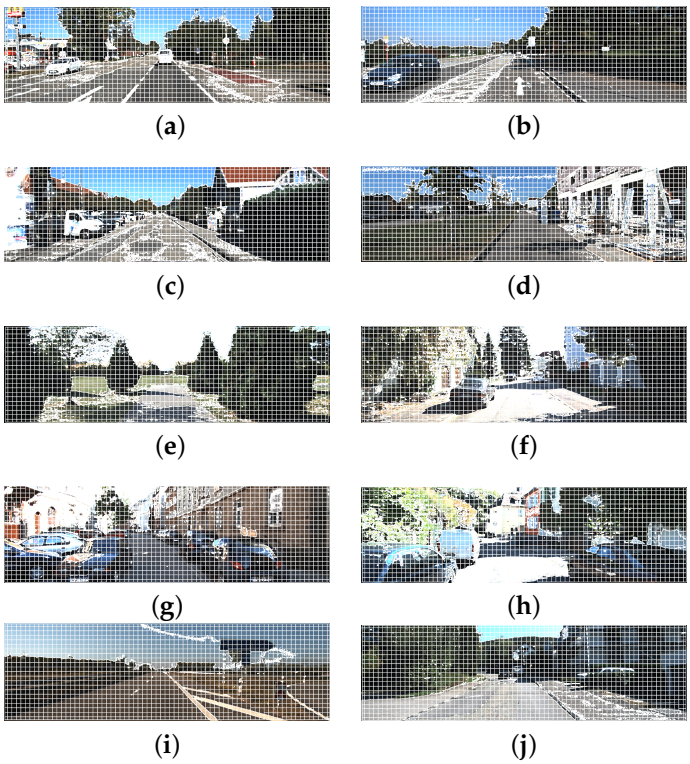


Figure 12. Superpixel segmentation: quantitative results for the KITTI dataset ($s = 4$). The subfigures (**a**–**j**) depict images from different sequences of the KITTI dataset in different urban environment contexts. With an increased number of superpixels, the method maintains accuracy while further enhancing regularity. Although beneficial for specific applications, the higher superpixel count may lead to increased computational demands for subsequent image processing tasks.

6. Application to Autonomous Vehicle Scenario

Our proposed superpixel segmentation algorithm holds considerable promise for real-time embedded applications, particularly in the realm of object recognition within autonomous vehicle scenarios. In the context of autonomous driving, the ability to recognize traffic signs, pedestrians, and other vehicles in real time is essential to ensure safe navigation. Using superpixels, our system can partition the image into coherent regions

that correspond to meaningful objects in the environment, thereby reducing computational complexity without compromising accuracy. In this study, we used the KITTI dataset, a widely recognized benchmark for autonomous driving research. The application of our superpixel algorithm enhances the detection of scene objects, including traffic signs and pedestrians, achieving greater efficiency. The segmentation process effectively groups pixels into regions that exhibit similar characteristics, such as color and texture, simplifying the tasks of object detection and classification. This capability is illustrated in Figure 13, which demonstrates the algorithm's effectiveness in identifying and delineating relevant objects within complex scenes.

The unprocessed image (Figure 13a) presents challenges such as cluttered backgrounds and varying lighting conditions, which lower the detection accuracy. In contrast, the segmented image (Figure 13b) detects key regions of interest, effectively separating the traffic sign from the background noise, and demonstrates several advantages in this context.

1. Improved Detection Accuracy: The segmentation process reduces the search space and minimizes false positives by focusing on regions with uniform characteristics, enabling the detection algorithm to operate more efficiently.
2. Enhanced Boundary Delineation: By ensuring high regularity and precision, the segmented image maintains accurate boundaries around the traffic sign, crucial for subsequent classification tasks.
3. Reduced Computational Load: The reduction in the number of processed regions results in faster detection times, a significant advantage for real-time embedded systems such as autonomous vehicles.



(**a**) Detection of traffic signs utilizing original data



(**b**) Detection of traffic signs utilizing our superpixel segmentation algorithm

Figure 13. (**a**) Detection of traffic signs utilizing original data. (**b**) Detection of traffic signs utilizing our superpixel segmentation algorithm. The green circles indicate the identified traffic lights, along with the detected traffic sign present in the scene. This detection is accomplished through a straightforward superpixel-based object recognition algorithm, as detailed in Appendix A.

7. Conclusions

Superpixel segmentation is designed to produce a coherent grouping of pixels. Despite the development of various superpixel segmentation algorithms, many existing methods suffer from low processing speeds and high demand for hardware resources, which can restrict their applicability in embedded systems. To address these challenges, we propose a novel superpixel segmentation algorithm that is implemented on an FPGA, offering significant improvements in algorithmic parallelization efficiency. This approach enables real-time

segmentation within a compact FPGA device, making our mathematical formulation ideal for intelligent embedded applications.

Compared to the current state of the art, our algorithm outperforms the previous approaches with regard to processing speed and hardware efficiency. The experimental results indicate that our solution achieves robust superpixel segmentation at speeds up to 11 times faster than those in previous studies while also demonstrating superior performance in hardware resource utilization. These features underscore the viability of our approach as a promising solution for modern embedded systems, particularly in applications such as autonomous driving and object recognition in real time. In addition, the integration of our FPGA-based superpixel segmentation algorithm not only enhances the capabilities of embedded systems but also opens new avenues for research and applications in the field of computer vision. Future work will focus on refining the algorithm and exploring its potential in diverse scenarios, including real-time video analysis and complex scene understanding.

Author Contributions: Conceptualization, Investigation: A.A.-G. Validation and Writing—Original Draft: A.M.S., J.A.d.J.O.-C., and J.A.O.T. Implementation for autonomous vehicles: M.P.P. and N.A.M.-N. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Acknowledgments: Acknowledgements to INAOE for supporting the development of this postdoctoral research under the supervision of Alejandro Medina Santiago (Researcher for Mexico); this work will strengthen Project 882 of Conahcyt.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. MATLAB Code for Traffic Sign Detection

The following code implements a simple traffic sign detection algorithm using superpixel segmentation and color-based feature extraction.

```
% Step 1: Convert the original image to HSV for color detection
hsvImage = rgb2hsv(imread('left/000007_10.png'));

% Define thresholds for the red color in HSV
% Adjust these thresholds based on the specific traffic signs in the dataset
mask = ((hsvImage(:,:,1) > 0.95 | hsvImage(:,:,1) < 0.05) & hsvImage(:,:,2) > 0.5
         & hsvImage(:,:,3) > 0.5) | ...
         ((hsvImage(:,:,3) > 0.95 | hsvImage(:,:,3) < 0.05) & hsvImage(:,:,1) > 0.5
          & hsvImage(:,:,2) > 0.5);

% Step 2: Apply the red color mask on the superpixel image
maskedSuperpixels = leftTurboPixels .* double(uint8(mask));

% Step 3: Identify connected components (superpixels) corresponding to red areas
labeledSuperpixels = bwlabel(maskedSuperpixels > 0);

% Step 4: Compute properties of the connected components
stats = regionprops(labeledSuperpixels, 'Area', 'Circularity', 'BoundingBox');

% Step 5: Filter out regions that are too small or do not resemble typical traffic sign shapes
Figure~(4);
imshow(imread('left/000007_10.png')); hold on;
for i = 1:length(stats)
```

```
    area = stats(i).Area;
    circularity = stats(i).Circularity; % 1 = perfect circle
    boundingBox = stats(i).BoundingBox;

    % Set thresholds for area and circularity (adjust based on dataset)
    if area > 200 && circularity > 0.6
        % Plot bounding box around detected traffic sign
        rectangle('Position', boundingBox, 'EdgeColor', 'g', 'LineWidth', 2);
        disp(['Traffic sign detected with area: ', num2str(area), ', circularity: ',
         num2str(circularity)]);
    end
end
```

References

1. Achanta, R.; Shaji, A.; Smith, K.; Lucchi, A.; Fua, P.; Süsstrunk, S. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell.* **2012**, *34*, 2274–2282. [CrossRef] [PubMed]
2. Budak, C.; Mençik, V. Detection of ring cell cancer in histopathological images with region of interest determined by SLIC superpixels method. *Neural Comput. Appl.* **2022**, *34*, 13499–13512. [CrossRef]
3. Quesada-Barriuso, P.; Blanco Heras, D.; Argüello, F. GPU accelerated waterpixel algorithm for superpixel segmentation of hyperspectral images. *J. Supercomput.* **2021**, *77*, 10040–10052. [CrossRef]
4. Shi, J.; Malik, J. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2000**, *22*, 888–905.
5. Salem, M.; Ibrahim, A.F.; Ali, H.A. Automatic quick-shift method for color image segmentation. In Proceedings of the 2013 8th International Conference on Computer Engineering & Systems (ICCES), Cairo, Egypt, 26–27 November 2013; pp. 245–251.
6. Csillik, O. Fast segmentation and classification of very high resolution remote sensing data using SLIC superpixels. *Remote Sens.* **2017**, *9*, 243. [CrossRef]
7. Mújica-Vargas, D. Superpixels extraction by an Intuitionistic fuzzy clustering algorithm. *J. Appl. Res. Technol.* **2021**, *19*, 140–152. [CrossRef]
8. Yang, F.; Sun, Q.; Jin, H.; Zhou, Z. Superpixel segmentation with fully convolutional networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 14–19 June 2020; pp. 13964–13973.
9. Ng, T.C.; Choy, S.K.; Lam, S.Y.; Yu, K.W. Fuzzy superpixel-based image segmentation. *Pattern Recognit.* **2023**, *134*, 109045. [CrossRef]
10. Giordano, D.; Murabito, F.; Palazzo, S.; Spampinato, C. Superpixel-based video object segmentation using perceptual organization and location prior. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 4814–4822.
11. Arbelaez, P.; Maire, M.; Fowlkes, C.; Malik, J. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2010**, *33*, 898–916. [CrossRef]
12. Yang, F.; Lu, H.; Yang, M.H. Robust superpixel tracking. *IEEE Trans. Image Process.* **2014**, *23*, 1639–1651. [CrossRef]
13. Tian, Z.; Zheng, N.; Xue, J.; Lan, X.; Li, C.; Zhou, G. Video object segmentation with shape cue based on spatiotemporal superpixel neighbourhood. *IET Comput. Vis.* **2014**, *8*, 16–25. [CrossRef]
14. Galasso, F.; Cipolla, R.; Schiele, B. Video segmentation with superpixels. In Proceedings of the Computer Vision–ACCV 2012: 11th Asian Conference on Computer Vision, Daejeon, Republic of Korea, 5–9 November 2012; Revised Selected Papers, Part I 11; Springer: Berlin/Heidelberg, Germany, 2013; pp. 760–774.
15. Cheng, J.; Liu, J.; Xu, Y.; Yin, F.; Wong, D.W.K.; Lee, B.H.; Cheung, C.; Aung, T.; Wong, T.Y. Superpixel classification for initialization in model based optic disc segmentation. In Proceedings of the 2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, San Diego, CA, USA, 28 August–1 September 2012; pp. 1450–1453.
16. Cheng, J.; Liu, J.; Xu, Y.; Yin, F.; Wong, D.W.K.; Tan, N.M.; Tao, D.; Cheng, C.Y.; Aung, T.; Wong, T.Y. Superpixel classification based optic disc and optic cup segmentation for glaucoma screening. *IEEE Trans. Med Imaging* **2013**, *32*, 1019–1032. [CrossRef] [PubMed]
17. Wang, M.; Liu, X.; Gao, Y.; Ma, X.; Soomro, N.Q. Superpixel segmentation: A benchmark. *Signal Process. Image Commun.* **2017**, *56*, 28–39. [CrossRef]
18. Felzenszwalb, P.F.; Huttenlocher, D.P. Image segmentation using local variation. In Proceedings of the 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No. 98CB36231), Santa Barbara, CA, USA, 23–25 June 1998; pp. 98–104.
19. Ghimire, M.; Regentova, E.; Muthukumar, V. A-SLIC: Acceleration of SLIC superpixel segmentation algorithm in a co-design framework. In Proceedings of the 17th International Conference on Information Technology–New Generations (ITNG 2020), Las Vegas, NV, USA, 5–8 April 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 663–667.

20. Ghaderi, A.; Ahlberg, C.; Östgren, M.; Ekstrand, F.; Ekström, M. Fp-slic: A fully-pipelined fpga implementation of superpixel image segmentation. In Proceedings of the 2022 25th Euromicro Conference on Digital System Design (DSD), Maspalomas, Spain, 31 August–2 September 2022; pp. 109–117.
21. Aguilar-González, A.; Arias-Estrada, M.; Pérez-Patricio, M.; Camas-Anzueto, J.L. An FPGA 2D-convolution unit based on the CAPH language. *J. Real-Time Image Process.* **2019**, *16*, 305–319. [CrossRef]
22. Aguilar-González, A.; Arias-Estrada, M. An FPGA stereo matching processor based on the sum of hamming distances. In Proceedings of the Applied Reconfigurable Computing: 12th International Symposium, ARC 2016 Mangaratiba, RJ, Brazil, 22–24 March 2016; Proceedings 12; Springer: Berlin/Heidelberg, Germany, 2016; pp. 66–77.
23. Geiger, A.; Lenz, P.; Stiller, C.; Urtasun, R. Vision meets robotics: The KITTI dataset. *Int. J. Robot. Res.* **2013**, *32*, 1231–1237. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.