sensors

MDPI

Article

Design of a Low-Power Embedded System Based on a SoC-FPGA and the Honeybee Search Algorithm for Real-Time Video Tracking

Carlos Soubervielle-Montalvo [1,*], Oscar E. Perez-Cham [1], Cesar Puente [1], Emilio J. Gonzalez-Galvan [1], Gustavo Olague [2], Carlos A. Aguirre-Salado [1], Juan C. Cuevas-Tello [1] and Luis J. Ontanon-Garcia [3]

[1] Facultad de Ingeniería, Universidad Autónoma de San Luis Potosí (UASLP), Dr. Manuel Nava No. 8, Zona Universitaria Poniente, San Luis Potosí 78290, San Luis Potosí, Mexico; oscar.cham@uaslp.mx (O.E.P.-C.); cesar.puente@uaslp.mx (C.P.); egonzale@uaslp.mx (E.J.G.-G.); carlos.aguirre@uaslp.mx (C.A.A.-S.); cuevas@uaslp.mx (J.C.C.-T.)
[2] Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE), Carretera Ensenada-Tijuana No. 3918, Zona Playitas, Ensenada 22860, Baja California, Mexico; olague@cicese.mx
[3] Coordinación Académica Región Altiplano Oeste, Universidad Autónoma de San Luis Potosí (UASLP), Carretera Salinas-Santo Domingo No. 200, Salinas de Hidalgo 78600, San Luis Potosí, Mexico; luis.ontanon@uaslp.mx
* Correspondence: carlos.soubervielle@uaslp.mx

Abstract: Video tracking involves detecting previously designated objects of interest within a sequence of image frames. It can be applied in robotics, unmanned vehicles, and automation, among other fields of interest. Video tracking is still regarded as an open problem due to a number of obstacles that still need to be overcome, including the need for high precision and real-time results, as well as portability and low-power demands. This work presents the design, implementation and assessment of a low-power embedded system based on an SoC-FPGA platform and the honeybee search algorithm (HSA) for real-time video tracking. HSA is a meta-heuristic that combines evolutionary computing and swarm intelligence techniques. Our findings demonstrated that the combination of SoC-FPGA and HSA reduced the consumption of computational resources, allowing real-time multiprocessing without a reduction in precision, and with the advantage of lower power consumption, which enabled portability. A starker difference was observed when measuring the power consumption. The proposed SoC-FPGA system consumed about 5 Watts, whereas the CPU-GPU system required more than 200 Watts. A general recommendation obtained from this research is to use SoC-FPGA over CPU-GPU to work with meta-heuristics in computer vision applications when an embedded solution is required.

Keywords: heterogeneous computing; meta-heuristic; video tracking; system-on-chip; field-programmable gate array; evolutionary computing; swarm intelligence; embedded system design; graphics processing unit; computer vision

check for updates

Citation: Soubervielle-Montalvo, C.; Perez-Cham, O.E.; Puente, C.; Gonzalez-Galvan, E.J.; Olague, G.; Aguirre-Salado, C.A.; Cuevas-Tello, J.C.; Ontanon-Garcia, L.J. Design of a Low-Power Embedded System Based on a SoC-FPGA and the Honeybee Search Algorithm for Real-Time Video Tracking. *Sensors* **2022**, *22*, 1280. https://doi.org/10.3390/s22031280

Academic Editors: Qiang Zhang and Yifeng Zeng

Received: 31 December 2021
Accepted: 2 February 2022
Published: 8 February 2022

1. Introduction

One of the goals of computer vision algorithms is the statistical analysis of the raw units of data contained in digital images or videos to allow the automated interpretation of visual information [1]. As in many other tasks, computer vision requires a massive number of operations on a massive amount of data. Video tracking is one of the computer vision problems that has become more relevant as a result of its application in various industries and technologies, such as robotics, unmanned vehicles, and automation, among others [2]. The video tracking problem essentially requires one to quantify the similarity of two ordered sets of data, with each component of these sets known as a pixel. Early video tracking approaches relied heavily on linear algebra and statistical examination to recognize patterns. Many different algorithms, colloquially known as video trackers, have been proposed in an attempt to solve the video tracking problem. Efforts to develop and improve video trackers are still ongoing; however, the problem remains open as perfect accuracy has not yet been achieved [3]. Additionally, each image in a digital video contains a massive amount of pixel data, resulting in high computational costs whenever they are inspected [4].

The purpose of video tracking, sometimes called object tracking, is to locate an object within a given image frame. It should be noted that the location of the object requires the detection of the same object in a previous frame [5]. Therefore, using a similarity measure, such as mean shift, normalized cross-correlation (NCC), regional mutual information (RMI), or zero mean normalized cross-correlation (ZNCC), is an elemental component to perform video tracking. Many of these techniques are still at the heart of newer proposals that use similar image matching criteria but which incorporate machine learning techniques, such as the support-vector machine of the Struck video tracker [6] or the decision tree of the tracker proposed by Asgarizadeh et al. [7] to choose between NCC and RMI as a similarity measure. The majority of today's most accurate video trackers, such as the SiamMask video tracker [8], are powered by convolutional neural networks [3]. However, these proposals have drawbacks that stem from the training phase, which introduces issues such as sampling bias, vulnerability to adversarial attacks, and a high demand for memory and computational resources to store and process considerable amounts of training data [9,10]. Another approach follows a symbolic learning approach, using an artificial dorsal stream as the backbone while incorporating genetic programming to build a robust machine learning paradigm [11]. The first results show the trustworthiness of results in such a computational visual process [12–14].

Although the video tracking community focuses on accuracy issues, other factors such as video tracker speed must also be considered [15]. For example, some studies in the literature have used tools such as heterogeneous computing architectures and population-based meta-heuristics to improve efficiency rather than efficacy [16,17]. A heterogeneous computing system is any computing system that coordinates two or more different types of processing elements. The processing elements that can be used are typically central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSP), or field-programmable gate arrays (FPGAs), but they can be applied to any other microprocessor design [18]. These systems are developed to meet the increasing demand for higher computational performance as well as lower energy consumption, total size or area of the hardware elements, and monetary prices [19,20]. The same motivations have driven manufacturers to innovate with the system-on-chip (SoC) paradigm. An SoC is any complex integrated circuit that encapsulates all of the elements of a fully functional system on the same chip [21]. A standard SoC includes a CPU, memory, hardware acceleration units, and peripheral interfaces.

Using different processing elements benefits different aspects of a problem. A CPU is better suited for control-intensive operations such as searching, parsing, and sorting, whereas a GPU is better suited for image processing or other data-intensive tasks [22]. The FPGA is a versatile tool, capable of performing both parallel and sequential tasks while consuming less power, allowing the system design to be customized according to the requirements of the problem to solve. On the other hand, it introduces new challenges, such as the requirement for hardware/software co-design methodologies [23]. Currently, CPU-GPU and CPU-FPGA platforms, particularly SoC-FPGA, are the most researched heterogeneous systems [20]. CPU-GPU-FPGA platforms are gaining attention as they enable developers to map computations to specialized processing units that can perform their tasks more efficiently. However, there are still issues with the CPU-GPU-FPGA platform, such as the increased task scheduling complexity, which is currently being investigated [24].

This work studies the benefits of the honeybee search algorithm (HSA) [25,26] in combination with the use of an SoC-FPGA platform to design and implement a low-power embedded system for real-time video tracking. HSA is a meta-heuristic that replaces the mechanical and meticulous inspection of all possible answers with a mindful search procedure that reduces resource waste. It is based on evolutionary algorithms that use biological evolution as a source of inspiration to perform intelligent directed searches over a given feasible space. These methods simulate how the fittest individuals survive and reproduce for several generations until a stopping condition is met [27]. HSA also belongs to the field of swarm intelligence meta-heuristics, which are inspired by the collective behavior of biological agents throughout their lifespan [28]. Swarm intelligence meta-heuristics are inspired by social animals such as ants, bees, fireflies, birds, and fish, including particle swarm optimization (PSO) [29] and artificial bee colony [30] and ant colony optimization [31]. Many of these meta-heuristics have been proven to help with real-world applications and benchmarking functions. However, no standardized decision criteria exist to select one over the others for any given problem.

1.1. Related Work

After the literature review, we found that there have been several research projects that explore the integration of population-based meta-heuristics and heterogeneous computing platforms applied to real-world problems. A common issue with swarm intelligence meta-heuristics implementations on CPU-GPU heterogeneous systems is that when global synchronization is required for certain activities, it cannot be achieved because GPU hardware is not designed to do so [32]. Depending on the model, GPU core synchronization is only possible between groups that belong to the same logic block. The synchronization issue significantly reduces the computational resources that can be used effectively to parallelize a swarm intelligence meta-heuristic using this type of heterogeneous system [33,34]. Commonly, the CPU and GPU are separate hardware products in a CPU-GPU heterogeneous system, whereas an SoC-FPGA is a single chip that contains the processing system (PS) and the programmable logic (PL). Because these processing elements are so close together, they can communicate more quickly while using less energy, with the advantage of using the custom advanced extensible interface (AXI) bus [35]. In other words, the internal communication of an SoC-FPGA device should outperform CPU-GPU communication via USB or PCI Express. Furthermore, the PS is typically built with ARM technology, which is a reliable and energy-efficient standard RISC architecture, widely used in embedded solutions centered on signal and image processing, such as video tracking [36].

Table 1 shows some works in the state of the art that belong to this research niche. A subtle insight is that most authors focus on the particle swarm optimization (PSO) meta-heuristic or variants of this approach. Other works use an FPGA device to study feasibility of their proposals, but this research attempts to contrast SoC-FPGA and CPU-GPU heterogeneous systems to find which is the best option in this particular setting. Additionally, the comparison with other state-of-the-art video trackers in terms of their speed and power consumption is also performed in order to highlight the advantages of our proposal. Another distinction with respect to our research is that evaluation is commonly performed using toy data, whereas this work uses the Amsterdam Library of Ordinary Videos (ALOV) benchmark, which provides 314 video sequences with annotated initialization and ground truth data [37]. These video sequences are arranged in 14 categories that pose different challenges for video trackers. The number and diversity of video sequences makes this benchmark a good alternative for testing the performance of video tracker proposals. The work of Morsi et al. [16] is particularly interesting because the application is video tracking, but the work uses only FPGA. Another interesting work is that of Nogueira and Barboza [17] because it studies SoC-FPGA and CPU-GPU heterogeneous computing platforms; however, the application is not specific as it focuses on continuous optimization problems.

In short, there are several opportunity areas. The literature has not sufficiently reported the advantages and disadvantages of CPU-GPU and SoC-FPGA platforms when used to implement meta-heuristics in computer vision problems. The main contribution of this work is the novel design of a low-power HSA-based embedded system developed on an SoC-FPGA platform for real-time video tracking, with the aim of determining whether a specific video tracker can be accelerated without a loss of accuracy. Moreover, the performance in terms of accuracy is evaluated using the ALOV benchmark in order to confirm that there are no negative effects compared to the employed base similarity measure (ZNCC). This work contributes with (1) the proposed workflow for the design of an embedded system based on an automaton that describes the behavior of a honeybee searching for food [38] and an SoC-FPGA platform; (2) the design, implementation, and evaluation of a low-power embedded system that performs real-time video tracking using the combination of the HSA meta-heuristic and an SoC-FPGA platform, which was designed from scratch using the co-design methodology [39]; (3) the comparison of our proposal with a CPU-GPU HSA-based video tracking system [34] in terms of speed, energy consumption, accuracy, as well as portability; and finally, (4) the contrast with other state-of-the-art video trackers in terms of speed and energy consumption to highlight the advantages of our proposal in these terms.

Table 1. Brief summary of the related work which focuses on other publications that deal with heterogeneous computing systems and meta-heuristics.

Source	Year	Meta-Heuristic	Heterogeneous Platform	Application
Palermo et al. [40]	2008	Discrete PSO	SoC-FPGA	Multi-objective Design, space exploration
Tsai et al. [41]	2010	DNA algorithm	SoC-FPGA	Fire extinguishing
Morsi et al. [16]	2013	PSO	FPGA	Structural similarity index for video tracking
Rodriguez and Moreno [42]	2015	GA (genetic algorithm)	FPGA	Motion estimation with particle filter
Elkhani et al. [43]	2018	Multi-objective binary PSO	CPU-GPU	Feature selection and classification
Perez-Cham et al. [34]	2020	HSA	CPU-GPU	ZNCC for video tracking
Nogueira and Barboza [17]	2020	GRASP (greedy randomized adaptive search procedure)	SoC-FPGA, CPU-GPU	Continuous optimization problems

1.2. Organization

The main sections of this paper are organized as follows. Section 2 goes into detail about the materials and methods used to enable other researchers to recreate and build on the results that are reported. Then, Section 3 delves into the specific experiments that were performed and the results that were obtained. The interpretation of those results, along with general observations, are discussed in Section 4. Finally, a summary of the highlights and future work is presented in Section 5.

2. Materials and Methods

This section details the hardware and software tools used to design and implement the proposed real-time video tracking embedded system. The vast majority of these materials are associated with the ZC706 SoC-FPGA evaluation board, which is used in this work. This evaluation board contains a Zynq-7000 SoC-FPGA [44], which has been successfully used for image processing [45], signal processing [46], Internet of Things [47], and other applications. The Zynq-7000 processing system (PS) is made up of a dual-core ARM Cortex-A9 processor, with a Kintex-7 FPGA [48] serving as the SoC programmable logic (PL). The main advantage of using an SoC-FPGA over an FPGA is the ease of use of a CPU to perform specific tasks, such as dealing with external memory devices and reading and preprocessing image files. The section will also provide details on the evaluation of the fitness function employed by the HSA meta-heuristic which is the ZNCC similarity measure; this fitness or objective function guides the exploration of the solution space to find a suboptimal approximation to the peak function value which maximizes the similarity between the juxtaposed image patches. The last section is dedicated to detailing the proposed workflow based on hardware/software co-design methodologies and focuses on how to take advantage of the HSA meta-heuristic and the SoC-FPGA platform.

2.1. Programmable Fabric

Given their notable emulation capabilities and availability, FPGA devices are currently the most popular type of programmable logic devices (PLDs). Using hardware description languages (HDLs) such as VHDL, these devices can be reconfigured to behave like a described hardware design [49]. The main top-level components of an FPGA are independently configurable and are included as follows: configurable logic blocks (CLBs), input/output blocks (IOBs), and switching matrices. CLB components implement internal logic behavior, IOB components receive and transmit external data, and all of these components are interconnected as specified by switching matrices. The CLB is composed of several smaller components that are used to implement custom logic. This section describes the primary components used by the ZC706 evaluation board's PL as the foundation for larger custom logic designs. The proposed custom system design makes use of a diverse set of Zynq-7000 PL components, the vast majority of which are categorized either as LUT (LookUp Table) or DSP48.

The LUT is the core component of the CLB, and each LUT is actually an array of one-bit multiplexers [50] with several one-bit inputs capable of implementing any arbitrary Boolean function with that number of inputs. In addition, each LUT contains a flip-flop that can be used to introduce sequential behavior if needed. The Boolean function of a given LUT is expressed as a truth table, which is stored in SRAM (static RAM) registers as needed. An FPGA has a large number of LUT components that allow for the implementation of complex combinational logic circuits using switching matrices. The DSP48 block is a physical ALU embedded in the fabric of Xilinx FPGA devices that is made up of three different blocks. This element consists of an add/subtract unit linked to a multiplier, which is linked to a final add/subtract/accumulate engine. The multiplier block can accept a single input value of up to 25 bits and a second input value of up to 18 bits. Given that multipliers consume a significant amount of programmable logic fabric resources (LUT components), using a DSP can result in area and power savings, while allowing the designer to use the programmable fabric for other tasks. The ZC706 board's Zynq-7000 SoC has a LUT count of 218,600 and 900 DPS48 blocks that can be used as the designer sees fit.

2.2. Soft Intellectual Property Cores

An intellectual property (IP) core is a reusable unit of logic, cell, or integrated circuit layout design that is the intellectual property of one party (such as Xilinx). Soft IP cores are frequently available as HDL files. Designers can easily implement those same cores on any FPGA to accelerate system design. This section goes over some of the IP cores used in the design of the proposed system described in Section 3.1. Xilinx Vivado 2017.3 was the IDE used in the system's development process, and it provided access to the used soft IP cores. Some vital Soft IP cores are those related to AXI communication between PS and PL.

Soft IP cores that perform relatively complex arithmetic operations, such as division or square root, are also important components. The system presented in this paper employs the CORDIC Soft IP core to compute square roots as needed to compute a given ZNCC value (Section 3.1). The CORDIC algorithm, also known as Volder's algorithm [51], is a method that can be implemented using low-level software or hardware that uses simple shift-add operations to perform a series of two-by-two matrix multiplications to compute trigonometric functions, hyperbolic functions, or square roots. However, some significant limitations of this resource include the inability to use the DPS48 blocks available in the SoC and the maximum input size of 48 bits. Xilinx's Divider soft IP core supports three different division algorithms: LUTMult, Radix-2, and High Radix [52]. The Radix-2 algorithm was chosen from among the three available algorithms because it returns the actual integer quotient of the division. LUTMult is a fast solution that takes advantage of the DSP48 blocks, but it consumes memory resources that should be used to store image data.

2.3. The Honeybee Search Algorithm Meta-Heuristic

The honeybee search algorithm (HSA) is similar to other honeybee-inspired meta-heuristics that mimic honeybees' search for food. The metaphor of flying individuals searching for food is useful for the general image processing problem because the search space is large in both cases and hints of the optimal solution are widely dispersed [38]. In that sense, HSA is heavily influenced by the dynamic flies heuristic of Boumaza and

Louchet [53], but with a greater emphasis on coordinated work. HSA differs from other approaches in that it is a hybrid proposal that combines techniques from swarm intelligence and evolutionary algorithms for real-valued multidimensional optimization problems [25].

Swarm intelligence and evolutionary algorithms developed independently, but they share similar terminology and can be generalized as population-based meta-heuristics [54]. These meta-heuristics are inspired by life and biological processes in their approaches to solving general optimization problems. In the terminology of population-based meta-heuristics, a specific solution is referred to as an individual. These methods are typically iterative, with an initial population of individuals that changes over time, guided by the fitness function. A generation is the term used to describe each iteration of this process. Several individuals compete metaphorically against one another, and it is determined which ones are better than others based on their fitness. The best individuals are rewarded, whereas the worst individuals are punished. This process continues for several generations until the population reaches a point where it is close to the optimal value. The $(\mu+\lambda)$ evolution strategy (ES) [55] is the fundamental evolutionary algorithm in HSA. This evolutionary algorithm has two populations: a parent population of μ individuals and an offspring population of λ individuals. The $(\mu+\lambda)$-ES is an elitist ES, which means that individuals from the parent population can be chosen to be part of the population of the next generation.

Exploration, recruitment, and foraging (Figure 1) are the three main stages of the original HSA proposal [26]. The exploration phase of the optimization procedure begins with an initial inspection of the search space using the described modified version of the $(\mu+\lambda)$-ES. The recruitment stage emulates how honeybees coordinate their efforts to find food sources. Real honeybees communicate with one another through their dance language, which allows them to spread very reliable and complex information [56]. These data are later used by their peers to make group decisions. Foraging is the final phase, which involves a series of $(\mu+\lambda)$-ES subphases but with smaller search spaces and a larger number of individuals. These new limits are based on the previous phase's resource planning and distribution. It was initially proposed that HSA should be run on a single CPU. However, Perez-Cham et al. [34] recently demonstrated that the same meta-heuristic could be executed on a CPU-GPU platform, with some benefits and drawbacks reported. This work applies the findings of that work to propose a workflow for designing another HSA implementation that uses an SoC-PFGA platform in an attempt to overcome the limitations observed with the restrictive CPU-GPU platform. Another recent publication [38] proposes an alternative representation of HSA using automata and formal languages theory. That representation is the basis for the proposed workflow (Section 2.5), facilitating the proper design process of the real-time video tracking embedded system (Section 3.1).

2.3.1. The Evolution Strategy of HSA

The differences between the HSA $(\mu+\lambda)$-ES and the canonical $(\mu+\lambda)$-ES are mainly related to the generation of the λ offspring population. HSA replaces the ordinary mutation operator of $(\mu+\lambda)$-ES, which is applied by adding normally distributed random values to the individual. Instead, HSA uses the polynomial mutation operator [55]. Another variation is that the common $(\mu+\lambda)$-ES does not use any crossover operator, whereas HSA uses the simulated binary crossover (SBX) operator [57]. Futhermore, to avoid local optimam HSA randomly generates some of the members of the λ population. These three subdivisions of the λ population are called α or mutant sons, β or crossover sons, and γ or random sons. To respect the common ES terminology α, β, and γ are also the scalar variables that determine the size of the populations in such a manner that $\lambda = \alpha + \beta + \gamma$. Depending on the specific problem, HSA may use the sharing operator of Goldberg and Richardson [58] that causes the effect of dispersion on the population instead of clustering. This is useful whenever a diverse population is required. It is important to mention that the populations of the exploration phase are labeled μ_e, λ_e, α_e, β_e, and γ_e. The foraging populations have their own names, μ_f, λ_f, α_f, β_f, and γ_f, to be discernible.

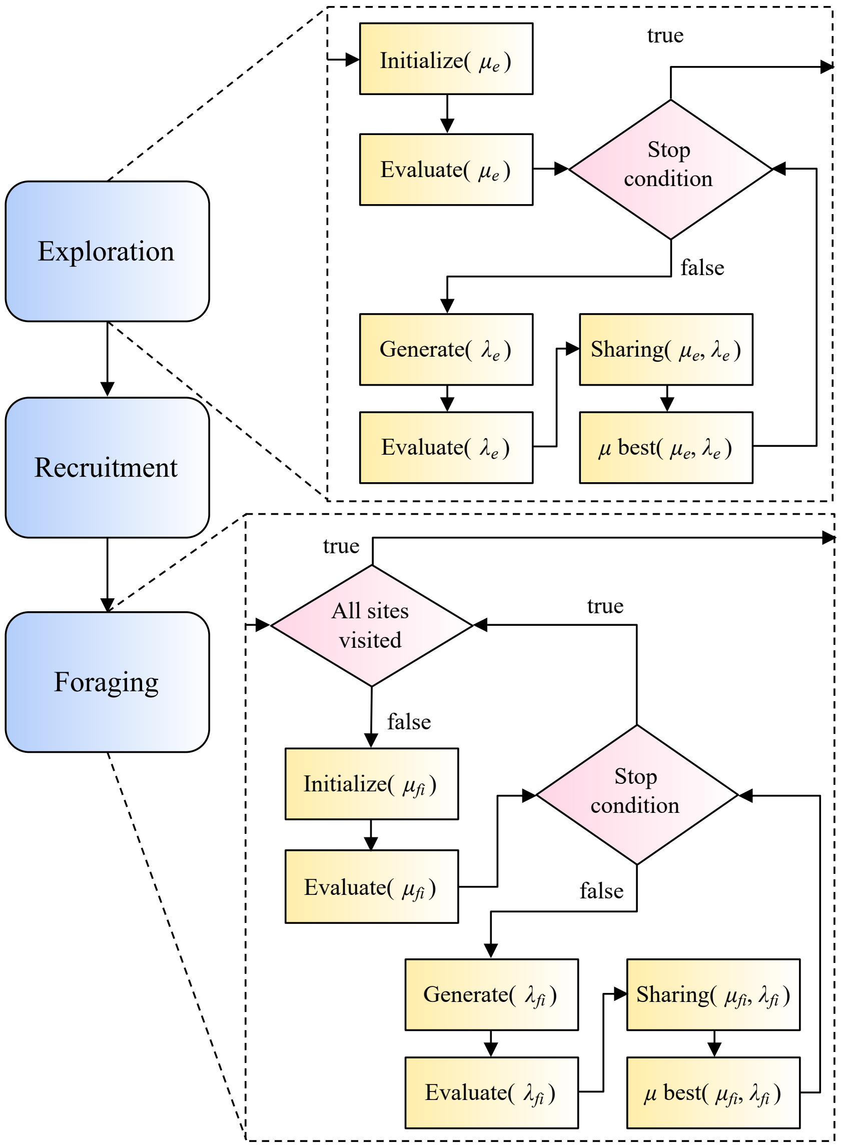


Figure 1. HSA has three main phases: exploration, recruitment, and foraging. The exploration and foraging phases are based on the canonical $(\mu + \lambda)$-ES but implement different evolutionary operators.

2.3.2. Polynomial Mutation

This kind of mutation operator applies variation using polynomial distribution perturbation [59]. The mutation operation modifies a parent solution x_a to generate a child solution x_b. Individuals are k-dimensional vectors, and the i-th dimension has a lower boundary L_i and an upper boundary U_i. Equations (1) and (2) are used to get the i-th component of individual x_b, where P_δ is obtained using a uniformly distributed random variable ρ that ranges from 0 to 1, and a user-defined non-negative value η_m that characterizes the probability distribution function.

$$x_{bi} = x_{ai} + (U_i - L_i) \cdot P_\delta \tag{1}$$

$$P_\delta = \begin{cases} (2\rho)^{1/(\eta_m+1)} - 1 & \text{if } \rho < 0.5 \\ 1 - [2(1-\rho)]^{1/(\eta_m+1)} & \text{otherwise} \end{cases} \tag{2}$$

2.3.3. Simulated Binary Crossover

The SBX operator [57] emulates the working principle of the single-point crossover operator on binary strings used by genetic algorithms. This operator receives two parent solutions x_a and x_b and uses them to create two children solutions x_c and x_d. Equations (3)–(5)

are used to generate the i-th component of k-dimensional vectors x_c and x_d, where P_β depends on a uniformly distributed random variable ρ that ranges from 0 to 1, and on a user-defined non-negative value η_c that characterizes the probability distribution function.

$$x_{ci} = 0.5[(1 + P_\beta)x_{ai} + (1 - P_\beta)x_{bi}] \tag{3}$$

$$x_{di} = 0.5[(1 - P_\beta)x_{ai} + (1 + P_\beta)x_{bi}] \tag{4}$$

$$P_\beta = \begin{cases} (2\rho)^{1/(\eta_c+1)} & \text{if } \rho < 0.5 \\ \left(\dfrac{1}{2(1-\rho)}\right)^{1/(\eta_c+1)} & \text{otherwise} \end{cases} \tag{5}$$

2.3.4. Recruitment Distribution

Equation (6) is used during the recruitment phase to distribute the individuals of the μ_f population into smaller groups μ_{fi} of size r_i each. There is one group of r_i individuals for every individual x_i of the μ_e population. The size r_i depends on the fitness value $f(x_i)$ of the individual x_i and the total sum of the fitness values of all the μ_e individuals. The individual x_i that has the greatest $f(x_i)$ will also have the greatest value of r_i. Metaphorically, the fittest exploration individual gets to recruit the largest number of peers from the foraging population to help in the exploitation of the resource that was found during the exploration phase.

$$r_i = \left\lfloor \frac{f(x_i)}{\sum_{j=1}^{\mu_e} f(x_j)} \times \mu_f \right\rfloor \tag{6}$$

2.3.5. Zero-Mean Normalized Cross-Correlation as Fitness Function

Zero-mean normalized cross-correlation (ZNCC) is a method used to determine how similar two ordered groupings of data are. This similarity grade is expressed as a scalar real number ranging from -1 to 1 [60]. If the ZNCC function returns 0, it signifies that there is no correlation or resemblance between the two sets of data. There is a degree of correlation or resemblance between the compared groups of data if the absolute ZNCC value is larger than zero. Negative ZNCC values, on the other hand, show that the relationship between the two sets of data is inversely proportional.

ZNCC is a widely used method in computer vision problems, and there have been recent efforts to simplify or speed up its calculation [61]. If a region within an image file is translated as a two-dimensional array of values, ZNCC can be utilized for video tracking or other computer vision applications. These values of the two-dimensional arrays are commonly grayscale pixels stored as integers ranging from 0 to 255 (8-bits). In this work, ZNCC is used as the HSA fitness function because ZNCC remains a reliable, deterministic, non-trained legacy tracker proposal that has been successfully interpreted as an objective function as it has a well-defined domain that can be translated into a search space [38].

Equations (7) through (12) detail the calculation of ZNCC, where (p, q) is a spatial coordinate within a given image frame I; t is an image template; $I_w \times I_h$ is the size of I; $\bar{t}$ is the mean of t; (u, v) is a spatial coordinate inside t with top left corner $(0, 0)$ and bottom right corner $(t_w - 1, t_h - 1)$; $\bar{I}(p, q)$ is the mean of a region of I that has the same size as t; ZCC is the zero mean cross correlation of an image patch; and SSE is the sum of squared errors of an image patch. ZNCC values of different coordinates are compared to find the spatial coordinate (p, q), where a given image template t is most certainly located within an image frame I. The original search approach is exhaustive, meaning the full search space is thoroughly inspected. However, the well-defined equations can be repurposed as a fitness function for two-dimensional individuals.

$$\bar{t} = \frac{1}{t_w \times t_h} \cdot \sum_{u=0}^{t_w-1} \sum_{v=0}^{t_h-1} t(u,v) \tag{7}$$

$$\bar{I}(p,q) = \frac{1}{t_w \times t_h} \cdot \sum_{u=0}^{t_w-1} \sum_{v=0}^{t_h-1} I(u+p, v+q) \tag{8}$$

$$\text{ZCC}(p,q) = \sum_{u=0}^{t_w-1} \sum_{v=0}^{t_h-1} [I(u+p,v+q) - \bar{I}(p,q)][t(u,v) - \bar{t}] \tag{9}$$

$$\text{SSE}_I(p,q) = \sum_{u=0}^{t_w-1} \sum_{v=0}^{t_h-1} [I(u+p,v+q) - \bar{I}(p,q)]^2 \tag{10}$$

$$\text{SSE}_t = \sum_{u=0}^{t_w-1} \sum_{v=0}^{t_h-1} [t(u,v) - \bar{t}]^2 \tag{11}$$

$$\text{ZNCC}(p,q) = \frac{\text{ZCC}(p,q)}{\sqrt{\text{SSE}_I(p,q) \cdot \text{SSE}_t}} \tag{12}$$

2.4. Evaluation with the Amsterdam Library of Ordinary Videos

The Amsterdam Library of Ordinary Videos (ALOV) was presented by Smeulders et al. [37] in order to allow the evaluation and comparison of video trackers. Three hundred and fourteen video sequences with annotated initialization and ground-truth data are considered part of this benchmark, many of which are also present on other benchmarks. These video sequences are classified into 14 different categories, each with its own set of problems for video trackers. Because of the large number and variety of video sequences, this benchmark is a suitable choice to test the robustness of video trackers. Another appealing feature of the ALOV benchmark is its concentration on common video sequences generated by ordinary camera equipment in real-world situations. Frequently, the video sequences of other benchmarks contain alternative pixel data, including thermal, infrared, or depth information, which present greater levels of challenge [3].

A video tracker's output is typically a rectangular area that encompasses the suspected position of the object of interest; this area is referred to as the video tracker's truth. However, the video tracker's truth does not have to be the actual location of the object of interest; that area is referred to as the ground truth. The amount of accuracy of the video tracker response can be geometrically translated as IoU $=$ intersection area $\div$ union area if the intersection and union areas of truth and ground truth can be measured. This notion of Intersection over Union (IoU) is utilized by many video tracking benchmarks, including the ALOV benchmark. A higher IoU value indicates greater accuracy, with a possible range from 0 to 1.

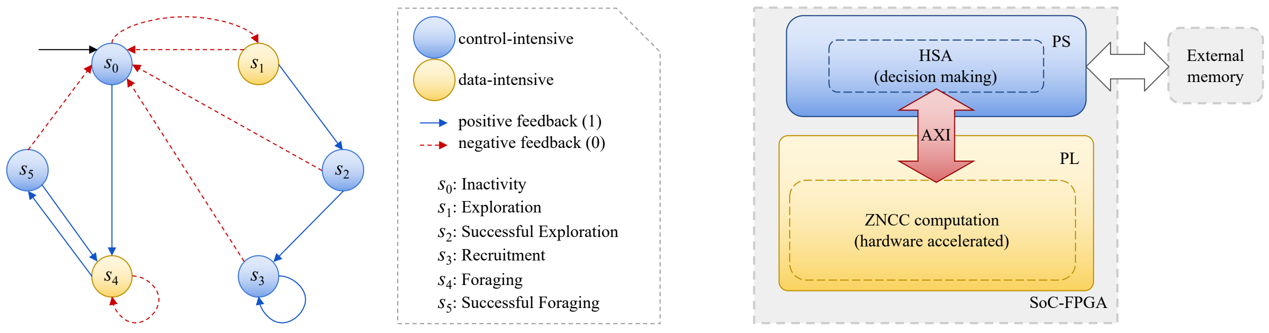
The ALOV benchmark recommends the detection of true positives and false positives in the evaluation process. This detection is based on a simple criterion, when IoU ≥ 0.5, the truth of a video tracker is considered a true positive. Given that this criterion evaluates only one frame of a given video sequence, another metric, the F-score, is used to measure the outcome of the entire video. The F-score F is calculated using Equation (13), where F_a is the number of true positives, F_b is the count of false positives, and F_c is the total of false negatives. Another ALOV benchmark recommendation is to use survival curve plots [62] to allow a simple visual inspection of the general trends observed in the F-score measurements of many video trackers with multiple video sequences. This same accuracy evaluation method is used in this paper.

$$F = \frac{F_a}{F_a + \frac{1}{2}(F_b + F_c)} \tag{13}$$

2.5. Proposed Workflow

When a designer proposes cooperating hardware and software components in a single design effort, a hardware/software co-design methodology can be used [23]. This methodology arises from the radically different tasks of designing and optimizing hardware and software. Hardware developers are taught to think in terms of parallel space decomposition, whereas software developers are taught to think in terms of sequential time decomposition. However, the terms hardware and software can mean different things in different contexts. As a result, some authors redefine the co-design methodology as application partitioning and design using fixed and flexible components [39]. The used co-design methodology combines traditional top-down and bottom-up methodologies because both are used at some point during the design process. A top-down approach starts with a high-level view of the application and then investigates how to improve its performance by utilizing specific resources [63]. On the other hand, in a bottom-up approach, the developer begins with the resources and investigates how to integrate them into a working system as efficiently as possible [64].

As previously stated in this paper, SoC-FPGA platforms are composed of two high-level components that are physically located on the same chip: the PL, which allows the emulation of a described electronic design; and the PS, which contains a dual-core ARM Cortex-A9 processor. Depending on the expected gain when parallelizing, we decided to perform certain HSA tasks in the PS and others in the PL in this case. The automaton (depicted in Figure 2a) and computational complexity analysis of Perez-Cham et al. [38] were critical tools that allowed the identification of the HSA meta-heuristic states or phases that benefit the most from hardware acceleration. Our conclusion was that the fitness function evaluation (ZNCC) is the part that incurs the most costs and has the greatest potential to be intuitively parallelized. As a result, this work proposes that the PS should perform HSA tasks related to intelligent coordinated decision making, whereas the PL should focus on ZNCC evaluation, corresponding to the exploration and foraging states (Figure 2b).



a) The automaton of the HSA meta-heuristic

b) System level abstraction, division of tasks between PS and PL

Figure 2. Proposed workflow to design the real-time video tracking system. (**a**) The automaton of Perez-Cham et al. [38] that formally describes the HSA meta-heuristic in an abstract non-architecture-specific manner. (**b**) System-level abstraction based on the HSA automaton that shows the division of tasks between PS and PL. The PS is in charge of control-intensive operations that are conducted by the HSA meta-heuristic (states s_0, s_2, s_3 and s_5), whereas the PL focuses on the acceleration of ZNCC via concurrency and parallel processing (states s_1 and s_4).

Following the establishment of a division of labor between PS and PL, the next priority is to design a hardware module capable of computing ZNCC based on input data, which consists of two small image patches. Given the ZNCC computation's sequential nature, this module should include a custom control unit and a custom datapath that performs the necessary arithmetic and logic operations while utilizing hardware concurrency whenever possible. Parallelization should also be possible through a coordinated array of ZNCC computation modules, each of which is dedicated to performing the same operations but on different input data (SIMD; single instruction, multiple data). The iterative design process for this ZNCC module employs both top-down and bottom-up approaches, adapting both the algebraic interpretation to the system's basic low-level components and vice versa. An abstract representation of the array of ZNCC modules is illustrated in Figure 3.

Finally, communication between PS and PL should be treated as a distinct issue. To solve this problem, a thorough understanding of Xilinx's AXI Bus solution is required. On SoC-FPGA platforms, communication between CPU and FPGA can be implemented in a variety of ways, but direct memory access (DMA) is best suited for data-intensive applications. When using DMA, the data transfer is handled by specialized hardware, whereas normal memory access is handled by the CPU [20]. This has two benefits: it speeds up the data transfer and frees up the CPU for other tasks. AXI CDMA is the most suitable solution and is thus used to communicate between the PS and PL components in the proposed system design. A detailed description of the implementation and operation of AXI CDMA is presented in Section 2.6.

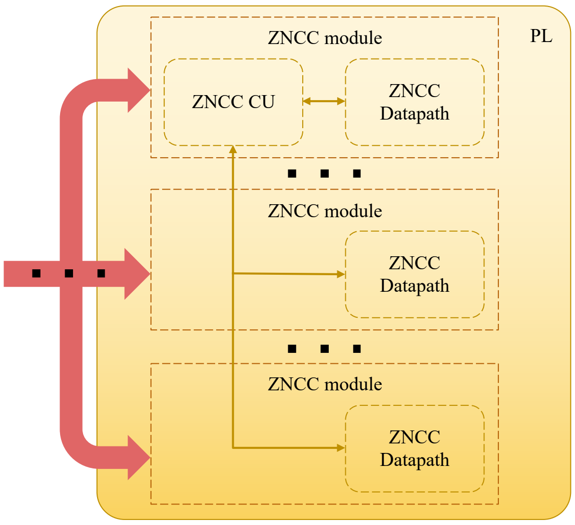


Figure 3. An illustration of how several ZNCC modules perform parallel processing, taking advantage of the capabilities of the PL. Each ZNCC module requires a datapath to perform operations simultaneously on different sets of input data, coordinated by a common control unit that oversees the execution of tasks following the SIMD parallel model.

The workflow described in this section served to guide the design process of the proposed embedded system, which is detailed in depth in the following section. As the niche is still being researched, the authors hope that other researchers will find this workflow proposal useful in proposing similar systems based on meta-heuristics and SoC-FPGA platforms.

2.6. Processing System and AXI Communication

The dual-core processor of the Zynq-7000 SoC can be programmed directly using assembly or low-level languages but this project uses a lightweight Linux operating system installation to allow the PS to serve as a development environment, along with compilation tools and useful software libraries such as OpenCV to easily manipulate image files. Furthermore, the operating system builds a logic layer that allows the use of board interfaces such as UART, USB, Ethernet, etc. A program running on Linux can interact with the physical memory that is directly used by the computer hardware, which is different to the virtual memory that is commonly used by programs. The interaction of a program with physical device memory is achieved through a file that is an image of the main memory, which can be located following the path /dev/mem, using the common root folder structure. In this case, the /dev/mem file is used to read and write data that are also manipulated by the PL, creating a bridge between PS and PL. AXI defines a protocol where several master and slave components interact using parallel, high-performance, synchronous, high-frequency transfer operations. Xilinx has adopted AXI4, AXI4-Lite, and AXI4-Stream [35] as the main communication interfaces that are used by their products. The PS can act as an AXI master or an AXI slave depending on the specific AXI peripherals that are implemented using the logic fabric.

As a reminder, the PS will focus on the activities of HSA that determine the intelligent behavior of the system, whereas the PL will focus on the acceleration of the fitness function (ZNCC). Communication between PS and PL is carried out through AXI interfaces of two different types: AXI GPIO and AXI CDMA. The simplest Xilinx AXI peripheral is the AXI GPIO (general purpose input/output) that directly reads or writes a register of 32 bits that are directly connected to physical hardware such as an output LED array, an input switch array, or other circuits implemented on the PL. AXI GPIO is used to send a 32-bit configuration word that is used to control the system design that is implemented using the PL. This configuration word is the length of the array of values that is sent as input. If the length value is zero, a global reset signal is fired.

ALOV image files are stored in JPEG format; however, they are read and converted to a one-dimensional array of grayscale unsigned integer values of 8 bits using software. These values are then transferred from the PS to the PL and stored using standard BRAM resources. The word size of the used BRAM is 32 bits, meaning that four grayscale values (of 1 byte each) can be written or read at once. The maximum capacity of BRAM resources is 8192 bytes (2^{13}) or 8 KiB and the data transfer is made using AXI CDMA blocks that read the image values from the PS and write them to the corresponding BRAM resource in the PL. The AXI CDMA (central DMA) peripheral can act as the PS AXI master, reading data from PS memory and writing it to other AXI slave peripherals' memories. The request to perform a data transfer contains the following data: clear/work/standby configuration word (32 bits), starting source address in the memory device (32 bits), starting destiny address in the BRAM implemented on the PL (32 bits), and the count of bytes to be transferred (32 bits). When the AXI CDMA block receives those simple orders, the PS acts as the master device. However, once the AXI CDMA block starts working, it acts as the master device with the permission to read PS device memory directly. Since AXI CDMA is only dedicated to transferring data, this hardware is noticeably faster than the PS itself.

3. Results

This section considers the final proposed embedded system design as a result, since it was generated through an iterative design process using the general guidelines that were established in the proposed workflow. The experimental results that validate the proposed system are also presented in this section. Some opportunity areas for improvement have been identified and are discussed below in the Discussion and Conclusions sections.

3.1. Proposed Embedded System Design

This section explains how the proposed embedded system is designed using the workflow detailed above, and also how it works in a modular way. The following sections detail the hardware design implemented using the PL component. As a reminder, the PL focuses on the evaluation of ZNCC, given that it was previously identified as the bottleneck, because it is the most computationally expensive task. A custom digital system design generally requires two top-level elements: the control unit and the datapath [65]. In order to facilitate the description of the proposed system, the description starts with the datapath, which is composed of elemental blocks that were created by combining the fixed components described in the previous sections. The average, SSE, and RSSE blocks are the primary blocks used to perform arithmetic and logic operations. These modules perform the necessary operations, but the sequential flow is controlled by finite state machines [66] implemented as control units. Multiple instances of these components can be implemented, allowing several ZNCC values to be computed at the same time. The main control unit, ZNCC CU (Figure 4) and the accumulation control Unit, CU (Figure 5) are defined below to explain the interrelations and data dependencies between the datapath elemental blocks.

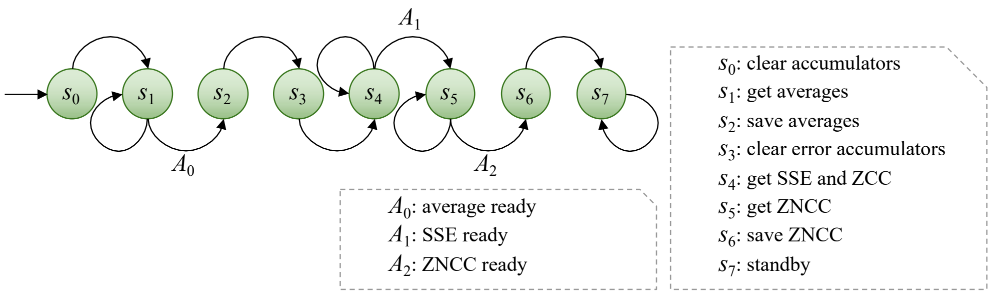


Figure 4. A visualization of the Moore machine of the ZNCC CU. This CU supervises the custom ZNCC module's proper functioning. It has control over another CU that manages accumulation and datapath elemental blocks.

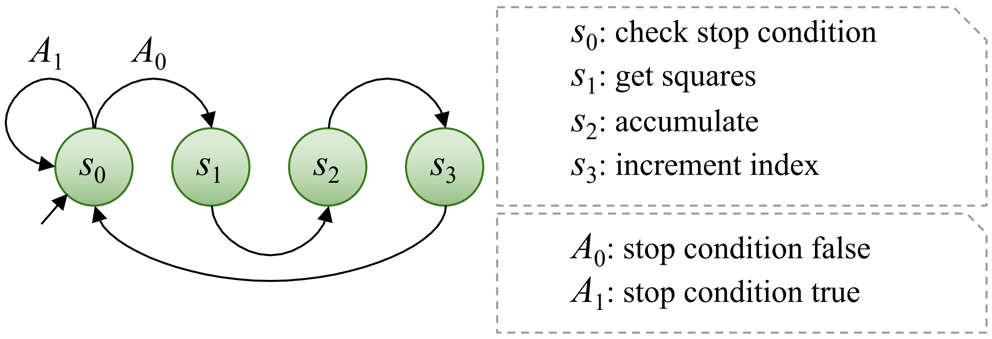
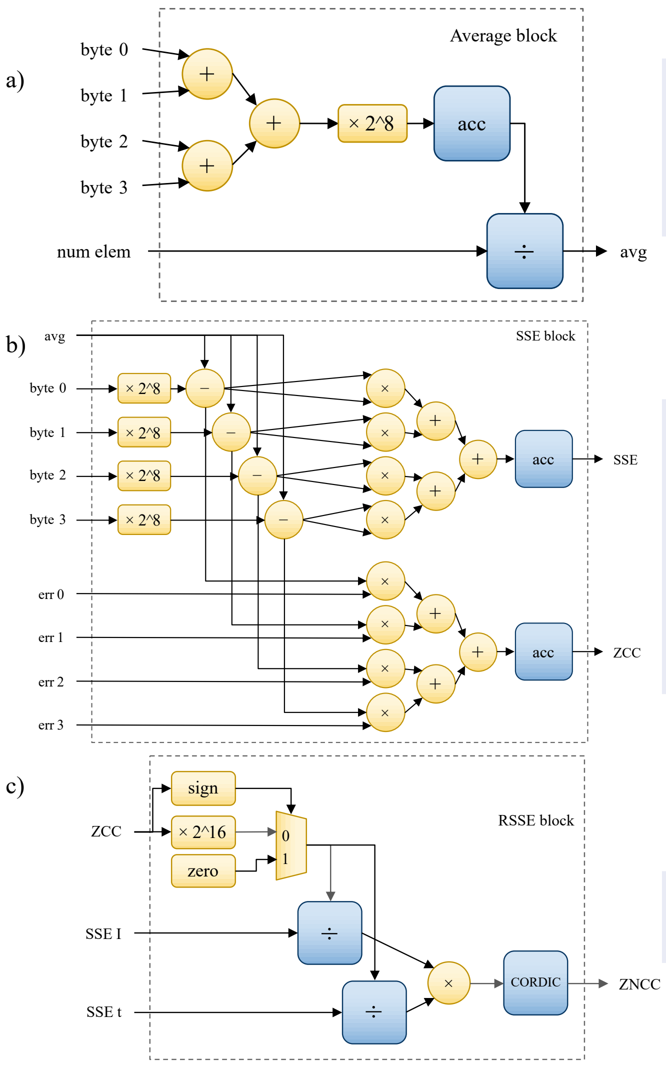


Figure 5. A Moore machine illustration describing the behavior of the accumulation CU. Summations constitute the vast majority of the sequential operations required to compute ZNCC, which is why the flow of those operations should be managed by means of a separate control unit.

3.2. Datapath

This section explains the main building blocks of each ZNCC module's datapath. The first block, the average block (Figure 6a), was designed using the bottom-up approach to calculate the average of a set of 8-bit positive integers. To adapt to the BRAM component that is used to store the array of input data, four input bytes are received at once. The resulting unsigned 8-bit integers are added using three simple binary adders. Then, using a hardwired shift left of 8 bits, the sum is multiplied by 2^8. This sum is the input to the main sequential accumulator, which follows the instructions of the accumulation CU (Figure 5) and ZNCC CU (Figure 4). The accumulator's 32-bit output is the dividend input of a sequential divider that employs the Radix-2 algorithm. The length of the array of numbers is the divisor input of the same divider block. The accumulation stop condition and the ZNCC CU provide the signal to begin working on the division. When the division is finished, an output signal is generated, which is the average done signal. The average is a 16-bit unsigned integer, stored in a register. This block's components are all implemented with the LUT programmable fabric.

The SSE block (Figure 6b) was also designed using the bottom-up approach. It takes four input bytes and applies the same treatment as the average block to produce four 16-bit unsigned integers. To obtain the individual error values, the previously obtained corresponding average is substracted from each of the integer values. It is worth noting that some of these blocks output error values (differences from the average), which can be used by other blocks to calculate ZCC. The block that computes both SSE and ZCC has eight 17-bit multipliers, four of which are used to calculate squared errors and the remaining four are used to cross multiply the errors. The accumulation CU (Figure 5) and the ZNCC CU (Figure 4) control the 48-bit accumulators in both variations. The DSP48 components are used to implement the multipliers, whereas the rest of the blocks are LUT-based.



$$\bar{t} = \frac{1}{t_w \times t_h} \cdot \sum_{u=0}^{t_w-1} \sum_{v=0}^{t_h-1} t(u,v)$$

$$\bar{I}(p,q) = \frac{1}{t_w \times t_h} \cdot \sum_{u=0}^{t_w-1} \sum_{v=0}^{t_h-1} I(u+p, v+q)$$

$$\text{SSE}_I(p,q) = \sum_{u=0}^{t_w-1} \sum_{v=0}^{t_h-1} [I(u+p,v+q) - \bar{I}(p,q)]^2$$

$$\text{SSE}_t = \sum_{u=0}^{t_w-1} \sum_{v=0}^{t_h-1} [t(u,v) - \bar{t}]^2$$

$$\text{ZCC}(p,q) = \sum_{u=0}^{t_w-1} \sum_{v=0}^{t_h-1} [I(u+p,v+q) - \bar{I}(p,q)][t(u,v) - \bar{t}]$$

$$\text{ZNCC}(p,q) = \sqrt{\frac{\text{ZCC}(p,q)}{\text{SSE}_I(p,q)} \cdot \frac{\text{ZCC}(p,q)}{\text{SSE}_t}}$$

Figure 6. This figure shows how the elemental blocks of the datapath are built from available soft IP cores and basic low-level and logic arithmetic blocks, each of which is implemented using the programmable fabric of the FPGA. (**a**) The average block; (**b**) the SSE block; (**c**) the RSSE block.

The RSSE block (Figure 6c) design process required a top-down approach. When working with integers, some complications arise, such as the loss of significant figures when the result of an operation has a decimal part. Floating-point operations, on the other hand, consume a lot of resources and are avoided in this design. This is why this system modifies ZNCC to always work with integers. The CORDIC block is used to compute the square root of a truncated integer given a 48-bit input integer. This work proposes using some simple algebraic manipulations to compute ZNCC more quickly without losing detail in the results; this is an example of how the function adapts to the available resources. Equation (14) demonstrates how to obtain the squared ZNCC value by performing two parallel divisions and one multiplication. ZCC values can be either positive or negative, which ultimately indicates if the detected correlation is inverse or not. This work proposes ignoring negative ZCC values, which are considered undesirable in the fitness function because they do not indicate a strong correlation of the compared image patches. This adjustment in the ZNCC calculation avoids confusion in this application and saves resources. A simple multiplexer is used to implement this, which outputs zero whenever a negative ZCC value is detected (Figure 6c).

$$\text{ZNCC}^2(p,q) = \frac{\text{ZCC}(p,q) \cdot \text{ZCC}(p,q)}{\text{SSE}_I(p,q) \cdot \text{SSE}_t} = \frac{\text{ZCC}(p,q)}{\text{SSE}_I(p,q)} \cdot \frac{\text{ZCC}(p,q)}{\text{SSE}_t} \quad (14)$$

To avoid the loss of resolution when the result is truncated, the ZCC input value of this block is multiplied by 2^{16} before calculating both divisions, as indicated in Figure 6c. The quotients of the dividing blocks are the inputs to a 24-bit multiplier, which is used to calculate the square of the ZNCC value, as indicated in Equation (14). A single CORDIC block starts working on the square root of the squared ZNCC value when the dividers emit a positive done signal. As a result, the output of the RSSE block is a positive integer ranging from 0 to 2^{16}, or 0 to 10,000 in hexadecimal numbers, where a value of 2^{16} means total similarity between both input images (t and I). At least one of these blocks is required for the system to function, but several can be used simultaneously to obtain a large number of ZNCC values.

3.2.1. Control Units

The main control unit is called ZNCC CU (Figure 4); it is in charge of the overall operation of the custom ZNCC module. It has authority over another CU that controls the accumulation and the datapath elemental blocks. The Moore machine used to describe the CU has eight states $S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$. The tasks that are performed in each of these states are: clear the average accumulators to start from zero (s_0); work to get averages using the accumulation CU (s_1); save the averages to use them later (s_2); clear the SSE and ZCC accumulators to start from zero (s_3); work to obtain the sums of squared error (SSE) and zero-mean cross-correlation (ZCC) values using the accumulation CU (s_4); work to compute the ZNCC value (s_5); save the ZNCC value (s_6); stand by and wait for a hard reset (s_7). The input alphabet A has three input characters: a signal that indicates that the average is ready (A_0), a signal that indicates that the SSE is ready (A_1), and a signal that indicates that the ZNCC value is ready (A_2).

Because several of the sequential operations required to calculate a ZNCC value are summations, the accumulation CU (Figure 5) was created to control their sequential flow. The same control unit is used to guide two different summation phases that are taken into account in the overall flow provided by the ZNCC CU. This control unit is used to perform sequential integer accumulation in two cases: to obtain sums that will be used to calculate averages, and to obtain SSE and ZCC values. All averages can be evaluated in parallel, but SSE and ZCC require averages to be computed first. As a result, the design takes into account two accumulation phases. The Moore machine that describes this CU has four states $S = \{s_0, s_1, s_2, s_3\}$, where s_0 is the state in which the stop condition is checked. Furthermore, during state s_0, the CU waits for the BRAM to deliver the data that are currently being requested. The data received from the BRAM are then multiplied to obtain the square in state s_1, if necessary. When the CU is in the s_2 state, the next step is to accumulate the value or the squared value. Finally, in state s_3, a signal is provided to indicate that the process's index should be incremented. There are only two input characters in the input alphabet A: the signal that indicates that the process is not complete (A_0) and the signal that indicates the opposite (A_1). These input characters are used to indicate when the stop condition has been met, which is that the index is greater than the length of the array. In general terms, the behavior of this CU is very similar to a for-loop in common programming.

3.2.2. System Overview

The average, SSE and RSSE blocks, which were discussed in previous sections, are the main blocks utilized to conduct arithmetic and logic operations. These modules carry out the necessary tasks, but the sequential flow is managed by finite state machines, which are implemented as control units. The PL can be used to implement several instances of these components, allowing multiple ZNCC values to be computed at the same time. The designed system can be characterized as an SIMD device because it performs the same actions on multiple sources of input data.

To compute four ZNCC values in parallel, the final iteration of the system (Figure 7) implements five average blocks, five SSE blocks, and four RSSE blocks. With the specific

SoC-FPGA that is employed (Zynq-7000 XC7Z045), a higher degree of parallelization is theoretically conceivable as the full implemented solution makes use of 40% of the available LUT programmable fabric and 7% of the DSP blocks. However, this version of the proposed system fits the resources of all the SoC-FPGA devices that contain an FPGA that belongs to the Xilinx Kintex-7 product family [48], making this design reproducible with a wider variety of devices.

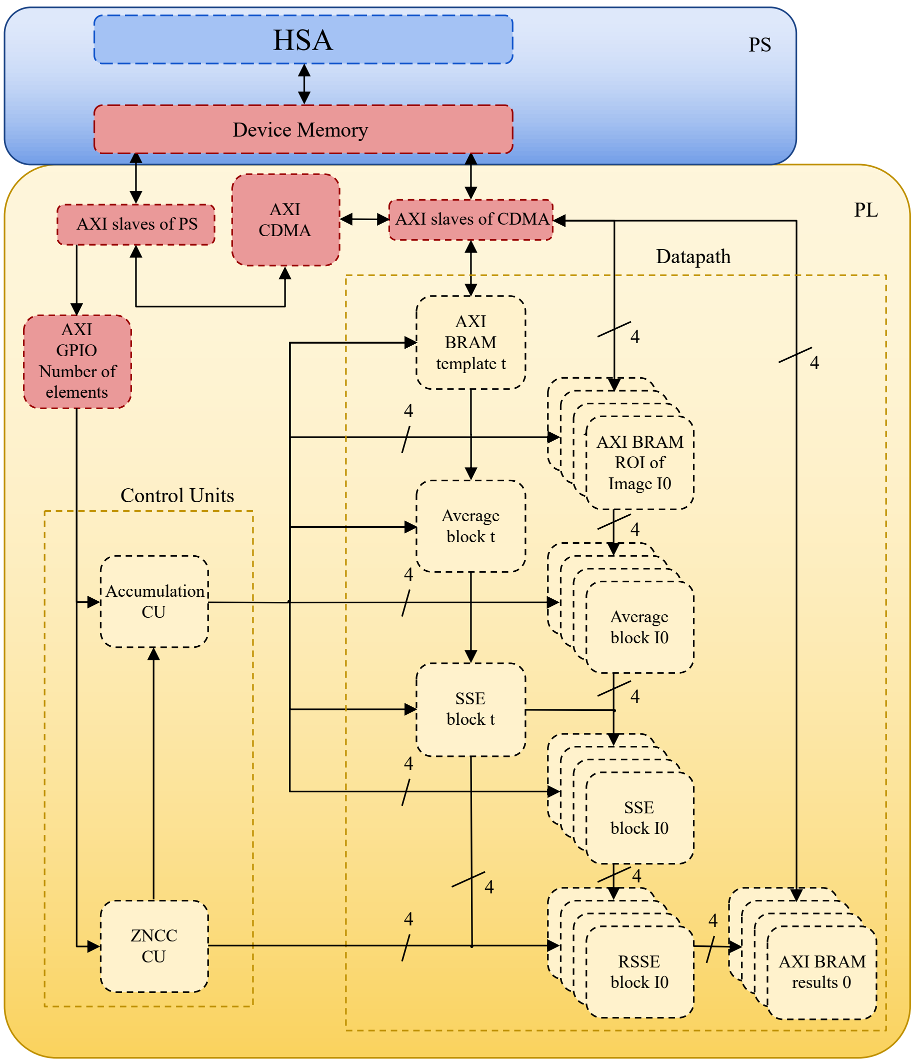


Figure 7. An illustration of the final interconnection of the ZNCC processing unit's top-level blocks. Average blocks, SSE blocks, RSSE blocks, the accumulation CU, and the ZNCC CU are the key blocks that were implemented utilizing the PL. The communication between PS and PL is implemented using AXI, where CDMA is used to transfer image data and GPIO is used to transfer simple control data.

3.3. Experiments

A full description of the system that was used in the reported experiments is provided in Section 3.1; the top-level overview is illustrated in Figure 7. As a reminder, this system is built upon the ZC706 Evaluation Board that contains a Zynq-7000 SoC-FPGA, specifically a XC7Z045 device. Most experiments are based on the data and methods of the ALOV benchmark or certain selected video sequences; the description of how ALOV is used for evaluation can be found in Section 2.4.

3.3.1. Calibration Test with a Static Image

By definition, real numbers ranging from −1 to 1 make up the range of values returned by the ZNCC function (Section 2.3.5). However, the custom hardware design proposed in this work modifies this range to make better use of the hardware resources. The following changes have been made: negative numbers are now ignored, the range has been adapted from 0 to 2^{16}, and the values are now discrete integer numbers; in other words, the result values are binarized. This section describes a simple test intended to confirm that the modifications have no negative effects on the accuracy of the ZNCC tracker.

The custom ZNCC unit was used in this test to perform the exhaustive search of a simple template within the same image frame where it was originally located. Figure 8a depicts the static image that was used. Given that the only point with 100 percent similarity is the same location as the initial object, the test is deemed successful. A closer look at the ZNCC values (Figure 8b) reveals that they are qualitatively correct because other similar instances of the object are also marked as bright spots.

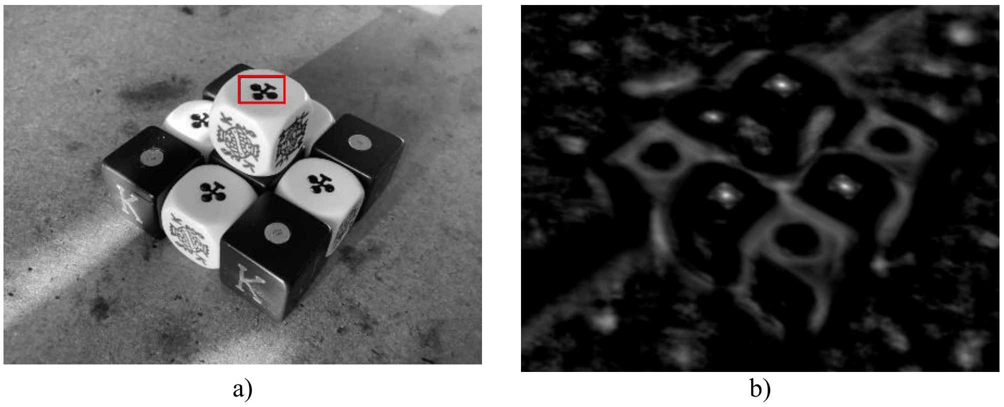


Figure 8. Visualization of the results of the calibration test. (**a**) A grayscale image with an instance of an ace of clover (object of interest) contained within a red bounding box. (**b**) A graphical representation of ZNCC values, with brighter spots representing the spatial location of the highest ZNCC values where similar objects are spotted.

3.3.2. Testing the SoC-FPGA ZNCC-Based System with One Video Using Exhaustive Search

Considering that the previous test demonstrated that the proposed ZNCC unit can find an object with 100% similarity, this test aims to confirm that the video tracking results are as accurate as those obtained with the plain CPU-GPU heterogeneous system of Perez-Cham et al. [34], which is also exhaustive. Due to the high time-costs associated with an exhaustive search, this test employed only one video. The chosen video is the twenty-third video from the ALOV benchmark's Light category, which follows a hamster ball as it rolls on the scene's floor (Figure 9). The obtained F-score was 0.975, which is identical to the result obtained using the CPU-GPU heterogeneous system. This demonstrates that the proposed ZNCC unit can identify a visually similar object even if the similarity is not perfect in the ZNCC scale.

Nevertheless, it was observed that when using a GPU, the execution times of the exhaustive search were reduced. The CPU-GPU heterogeneous system took 0.0503 s per frame on average to process this video, whereas the SoC-FPGA with the proposed configuration had a time-cost of 5.0004 s per frame. The superiority of the CPU-GPU heterogeneous system is expected in this setting, given that the GPU is the ideal architecture for a naive exhaustive search and that the GPU's full capacity is used in that case. The following examinations concentrate on the effects of the HSA meta-heuristic.

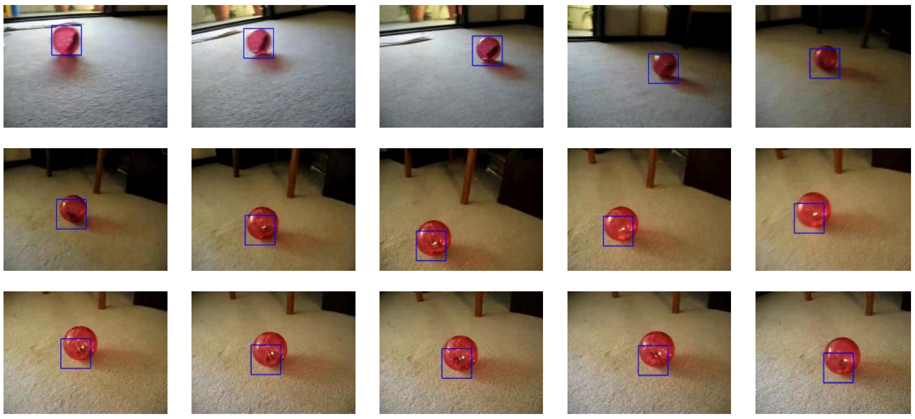


Figure 9. This figure displays some of the video frames that show how the tracker follows the object of interest (hamster ball). This test was made using the proposed embedded system to determine the feasibility of using it to accelerate an exhaustive search.

3.3.3. CPU-GPU versus SoC-FPGA Using HSA

The SoC-FPGA heterogeneous system was used to process 309 ALOV benchmark videos (Section 2.4). As explained in Section 3.1, the PL computes ZNCC to be used as a fitness function, and the PS is in charge of the HSA meta-heuristic. The accuracy comparison using the F-score (Figure 10a,b) shows that there is only a minor difference between using the SoC-FPGA heterogeneous system and the CPU-GPU heterogeneous system. However, the difference is only 6.6 percent of the total standard deviation, which is insignificant. A Student's *t*-test of two samples was used to compare the mean accuracy of CPU-GPU against SoC-FPGA. The resulting *p*-value of 41.19% (considerably greater than 5%) confirms the null hypothesis (means are the same). This indicates that the results of the implemented video tracking embedded system do not show any significant differences in accuracy.

The work of Perez-Cham et al. [34] provides a computational complexity analysis that provides evidence that the size of the object of interest in pixels has a significant effect on time-costs whenever HSA is used; this fact influences the following statements about the observed time-costs. In terms of time-costs, the SoC-FPGA outperformed the CPU-GPU (Figure 10c,d). The average time per frame (measured in seconds per frame) obtained with the SoC-FPGA was 0.1697, which is less than the average of 0.3729 obtained with the CPU-GPU heterogeneous system. Furthermore, the standard deviation of the time per frame measurements obtained using the SoC-FPGA was 0.1199, whereas the standard deviation of the CPU-GPU heterogeneous system was 0.3461, indicating that the SoC-FPGA was approximately 2.8 times more stable.

Using the SoC-FPGA increases the speed of this video tracker proposal and enables real-time processing with specific frame sizes (Figure 10e). Thirty-two videos (10.19%) from the ALOV dataset were processed in real-time; the videos were those where the size of the template was less than 3800 pixels. As a reminder, 30 frames per second was previously defined as the cutting edge of real-time video tracking, at least for this work. Furthermore, the SoC-FPGA consumed less power (Figure 10f). According to the hardware specifications, the CPU-GPU heterogeneous system consumed approximately 40 times more power than the SoC-FPGA heterogeneous system. This suggests that the use of an SoC-FPGA heterogeneous system combined with HSA is an efficient way to accelerate a video tracker.

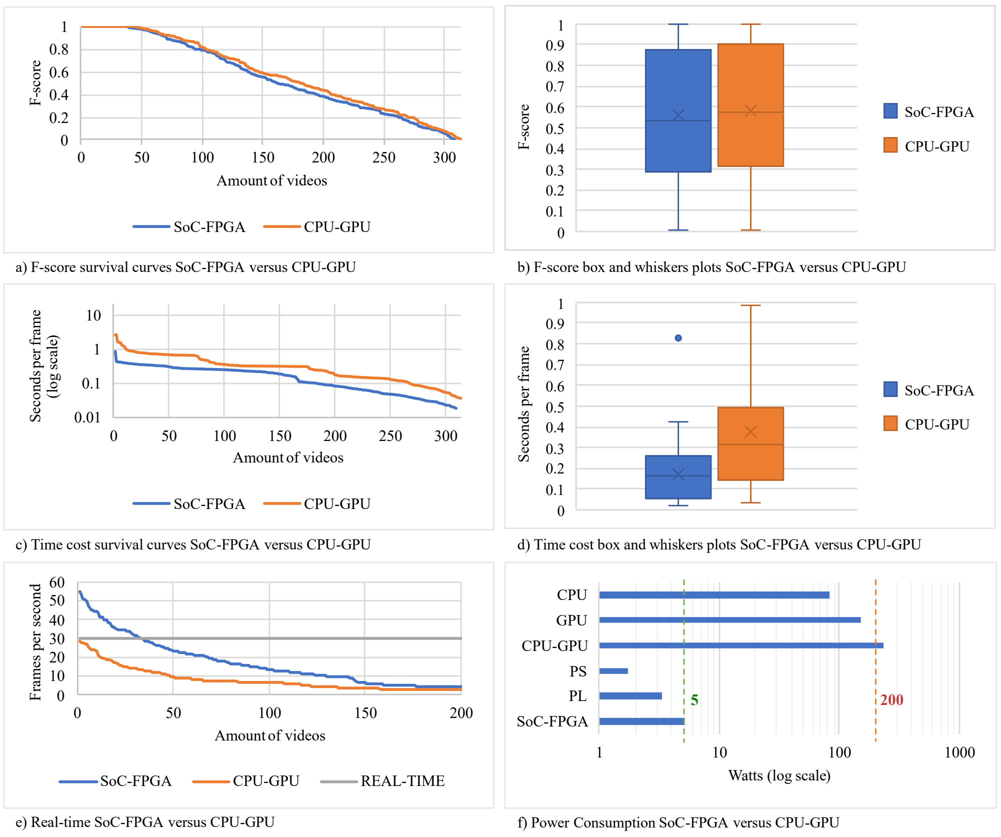


Figure 10. The compared video tracking systems use the same similarity measure (ZNCC) and the same meta-heuristic (HSA). The main difference between these proposals is the platform used to build the systems (SoC-FPGA or CPU-GPU). (**a**) SoC-FPGA versus CPU-GPU in terms of accuracy using survival curves. (**b**) SoC-FPGA versus CPU-GPU in terms of accuracy using box-and-whisker plots. (**c**) SoC-FPGA versus CPU-GPU in terms of time-costs using survival curves. (**d**) SoC-FPGA versus CPU-GPU in terms of time-costs using box-and-whisker plots. (**e**) SoC-FPGA versus CPU-GPU in terms of real-time video tracking. (**f**) SoC-FPGA versus CPU-GPU in terms of power consumption.

3.3.4. Comparison Against State-of-the-Art Trackers in Terms of Time-Costs

The experimental results of Perez-Cham et al. [34] provide evidence that the measured accuracy of the CPU-GPU system is no match for recent state-of-the-art video tracker proposals such as Struck and SiamMask. This is related to the underlying similarity measure that is used (ZNCC). However, it was important for the development of this work to maintain a certain degree of complexity as the integration of SoC-FPGA and HSA was not a trivial matter. Additionally, the simplicity of using ZNCC as a similarity measure allowed a useful and thorough cross-platform study, which was already expounded in the previous section. This has been the primary goal of the ongoing research project that envisions a unified methodology to integrate meta-heuristics and SoC-FPGA platforms, which remains an unexplored field.

With that in mind, it would be redundant to report that the low-power embedded system design for real-time video tracking presented in this work is not a suitable competitor in terms of accuracy, as it was demonstrated that the CPU-GPU system and the SoC-FPGA system deliver the same levels of accuracy. However, it should be noted that the new proposal is in fact close to becoming a strong contender in terms of time-costs, as shown in Figure 11. It should be noted that the time-costs of the original proposal based on CPU-GPU are substantially higher than those of the newer SoC-FPGA-based proposal and this trend should continue. Another important factor is power consumption and portability. Although Struck and SiamMask deliver outstanding accuracy, it is only possible to meet their energy and processing demands by using recent high-end mainstream CPU-GPU devices and computers.

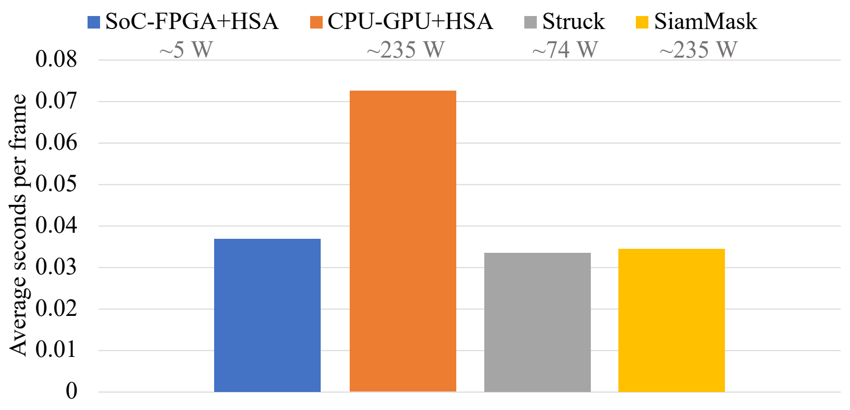


Figure 11. Comparison of the SoC-FPGA system against state-of-the-art trackers in terms of time-costs. The results of the CPU-GPU system are shown for contrast. Labels provide information about the estimated power consumption of the hardware that is typically used in each proposal. Note that the SoC-FPGA system is the only proposal that meets low-energy requirements.

4. Discussion

The scope of this project was to demonstrate that it is possible to develop an embedded solution for real-time video tracking with low power consumption and that it obtains reliable results, combining the benefits of the SoC-FPGA platforms and bioinspired meta-heuristics, which have been described in detail in previous sections of this work. The proposed system design uses 40% of the LUT (87,440) and 7% of the DSP48 (63) resources that are available on the ZC706 evaluation board, which allows scalability and portability, using similar platforms. In future works, we will consider redesigning the system to fit the full extension of resources of the ZC706, focusing on different aspects such as higher speed or even higher precision. Table 2 shows the properties of Zynq-7000 SoC-FPGA devices that include a Kintex-7 FPGA [48], to illustrate the posibility of portability and scalability employing different SoC-FPGA evaluation boards.

Table 2. Properties of Zynq-7000 SoC-FPGA devices that include a Kintex-7 FPGA

Device Name	Part Number	LUT Count	DSP Slices	Conforms to System Design
Z-7030	XC7Z030	78,600	400	Yes **
Z-7035	XC7Z035	171,900	900	Yes
Z-7045 *	XC7Z045	218,600	900	Yes
Z-7100	XC7Z100	277,400	2020	Yes

* This is the specific SoC-FPGA that was used in the experiments. ** The current design uses an LUT count of 87,440 and 63 DSP slices; an adaptation using more DSP slices instead of LUT components is feasible due to the excess of unused DSP slices.

A common trend in embedded system design that employs FPGA technology is to select relatively simple computer vision algorithms, given the difficulty of translating them to hardware. A ZNCC tracker was selected to be used in this work because it has a low complexity, which allows a more feasible and functional design and implementation in the SoC-FPGA platform. As explained in Section 3.3.4, ZNCC was fundamental to this study due to its low complexity, which permitted cross-platform testing. Additionally, ZNCC reliably detects similarity and remains a feasible building block for mature video tracking solutions. Moreover, the ALOV benchmark [37] which was used initially considered ZNCC one of the contenders, given that it is a legacy tracker, and this allowed us to compare it against the original CPU approach. The accuracy levels that have been attained using recent video trackers such as Struck [6] and SiamMask [8] are superior to that of ZNCC.

However, the proposed system obtains an average real-time result similar to those obtained for these state-of-the-art trackers, but with the advantages of lower power consumption and portability.

The precise definition of real-time image processing varies from 10 to 240 frames per second [15,67]. However, this work considers 30 frames per second as the minimum speed of a real-time tracking system as this measure is based on human visual perception. The speed of 30 fps can be achieved using the combination of HSA and SoC-FPGA, as shown in Figures 10b and 11. However, it is only possible when the object of interest is contained on a rectangle with a size of 3800 pixels or lower. The only relationship we observed between videos that were processed in real-time and their category is that these videos were those in which the object was relatively small compared to the size of the frame, particularly in surveillance cameras. In embedded system design, it is very important to focus efforts on a well-defined specific application. According to the evaluation carried out with the ALOV benchmark, we concluded that the ideal use of the proposed system is in low-resolution videos used for surveillance, where it is intended to follow people or vehicles which have a small size with respect to the size of the frame, which is observed when the camera it is found at high points or far away.

The CPU-GPU architecture has some limitations, including the high time-costs of communication, the limited coordination of GPU components, and the fixed nature of the CPU-GPU architecture. The implemented solution based on CPU-GPU and HSA used only a fraction of the available GPU cores, and most of them remained idle. The CPU-GPU heterogeneous system was created to accelerate massive rendering operations. According to the experiments presented in this work, CPU-GPU is better suited to performing exhaustive searches. The SoC-FPGA allows a greater control of the datapath and control units to perform image multiprocessing. The modules of the atomic operations may be used as required by the specific problem. In the case of the fitness function (ZNCC), the problem was reassessed to fit the available resources in a bottom-up fashion. One of the few disadvantages of using SoC-FPGA is that a complex iterative co-design methodology is required, which demands a deep understanding of different computer architectures (dual core ARM, FPGA, IP cores, DPS, etc.) to find an optimal design that is effective, energy-efficient, and which requires less computational resources and processing time. However, introducing HSA simplifies the hardware design process. Given that HSA does not explore the full search space, it is not strictly necessary to transfer the full frame from PS to PL. In that sense, HSA reduces the demand on memory resources.

5. Conclusions

In summary, this work presented the novel design, implementation, and evaluation of a low-power embedded system based on an SoC-FPGA platform and the HSA meta-heuristic for real-time video tracking. The main conclusions of this work are listed below.

1. An original workflow was proposed for the design of a low-power embedded system for real-time video tracking, based on an automaton that describes the behavior of a honeybee searching for food [38] and an SoC-FPGA platform. The workflow described in Section 2.5 served to guide the design process of the proposed embedded system. As the niche is still being researched, we hope that other researchers will find this workflow proposal useful in order to suggest similar systems based on meta-heuristics and SoC-FPGA platforms. It is useful to identify which parts of the meta-heuristic are control-intensive and which ones are data-intensive to identify the labors of PS and PL.
2. A novel design, implementation, and evaluation of a low-power embedded system that performs real-time video tracking by combining HSA meta-heuristics and an SoC-FPGA platform was presented. Several benefits were observed using HSA in combination with SoC-FPGA for video tracking. The SoC-FPGA allows a greater control of the modules of the atomic operations. In the case of the fitness function (ZNCC), the problem was reassessed to fit the available resources in a bottom-up fash-

ion. The time-costs are lower using an SoC-FPGA, which makes real-time processing possible. Furthermore, SoC-FPGA makes it possible to process a greater frame size in real time. Additionally, SoC-FPGA allows noticeably lower power consumption than CPU-GPU platforms and a greater portability. The experiments demonstrated that HSA can successfully be used to accelerate ZNCC for video tracking using SoC-FPGA without negative effects on accuracy.

3. The comparison of our SoC-FPGA HSA-based proposal with a CPU-GPU HSA-based video tracking system [34] in terms of speed, energy consumption, accuracy, as well as portability, allowed the identification of the limitations of the CPU-GPU platform in this context. These limitations were the high time-costs of communication, the limited coordination of GPU components, and the fixed nature of the CPU-GPU architecture. We recommend using CPU-GPU over SoC-FPGA only if the problem requires an exhaustive search and the solution does not require portability and consider using a meta-heuristic over a GPU whenever possible. The greatest reduction in time-costs was observed when HSA was used in combination with SoC-FPGA.The results of the evaluation provide evidence that the combination of SoC-FPGA platforms and meta-heuristics is promising as it enables the creation of portable, energy efficient, fast, and effective systems.
4. The results of the comparison with other state-of-the-art video trackers (Struck and SiamMask) showed that our proposal has the advantages of lower power consumption and portability, while maintaining similar processing speeds. On the other hand, Struck and SiamMask deliver outstanding accuracy, but they require high-end mainstream CPU-GPU devices and computers with high energy consumption. In this sense, the proposals of this work demonstrate that studying how to properly exploit the efficiency of the SoC-FPGA platforms in combination with meta-heuristics will bring substantial benefits to video tracking, other computer vision applications, and computational optimization in general.

Taking those findings into account, we propose the following future work to describe the direction of our research, considering the corresponding novel design, implementation, and evaluation tasks.

1. To improve the system that was designed to use the full capacity of the SoC-FPGA. The current proposal uses 40% of the LUT components, and 7% of the available DSP blocks. Additionally, we aim to exploit the possibility of reconfiguration, which allows the designer to propose many different designs that solve the same problem but with varying degrees of sequential and parallel behavior.
2. To verify whether HSA may be implemented using the PL of the SoC-FPGA. The current proposal uses the PL to compute the fitness function, but the general decision-making process is executed using the PS. Further experiments should be performed to find the advantages and disadvantages of using the PL to run HSA.
3. To use other fitness functions to replace or complement ZNCC. The canonical ZNCC tracker is currently not a viable contender against state-of-the-art trackers in terms of accuracy. However, it served as a starting point to study the advantages of using HSA and different heterogeneous systems for video tracking given its relative simplicity in comparison to newer proposals such as Struck [6] and SiamMask [8].
4. To use the combination of HSA and SoC-FPGA platforms in other CV applications. The results of using HSA for video tracking showed positive results. This motivates us to study the effect of HSA on other CV applications or on specific variations of the tested problems, for example, in face tracking and detection, or tracking based on infrared image data.

Author Contributions: Conceptualization, C.S.-M. and O.E.P.-C.; methodology, C.S.-M., O.E.P.-C. and C.P.; software, O.E.P.-C., C.S.-M. and C.P.; validation, E.J.G.-G., G.O. and C.P.; formal analysis, C.P., O.E.P.-C. and E.J.G.-G.; investigation, O.E.P.-C., C.A.A.-S. and J.C.C.-T.; resources, E.J.G.-G., G.O. and L.J.O.-G.; data curation, O.E.P.-C., J.C.C.-T. and C.A.A.-S.; writing—original draft preparation, C.S.-M. and O.E.P.-C.; writing—review and editing, E.J.G.-G., G.O. and C.A.A.-S.; visualization, G.O., J.C.C.-T. and L.J.O.-G.; supervision, C.S.-M.; project administration, C.S.-M. and C.P.; funding acquisition, C.S.-M., L.J.O.-G. and C.A.A.-S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by SEP-PRODEP, through project 511-6/18-8316, CA Code UASLP-CA-268. In addition, the second author, O.E.P.-C. thanks CONACYT for its support through grant No. 25096.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The ALOV dataset used in this work is available on http://alov300pp.joomlafree.it/.

Acknowledgments: The first author, C.S.-M. wishes to thank Liboria Montalvo-Castillo for her support in his process of becoming a researcher.

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

Abbreviations

The following abbreviations are used in this manuscript:

ALOV	Amsterdam Library of Ordinary Videos
ARM	Advanced RISC Machine
AXI	Advanced eXtensible Interface
CDMA	Central DMA
CLB	Configurable Logic Block
CU	Control Unit
CPU	Central Processing Unit
DMA	Direct Memory Access
DSP	Digital Signal Processor
ES	Evolution Strategy
FPGA	Field-Programmable Gate Array
GA	Genetic Algorithms
GPU	Graphics Processing Unit
GRASP	Greedy Randomized Adaptive Search Procedure
HDL	Hardware Description Languages
HSA	Honeybee Search Algorithm
IOB	Input/Output Blocks
LUT	LookUp Table
NCC	Normalized Cross-Correlation
PL	Programmable Logic
PLD	Programmable Logic Devices
PS	Processing System
PSO	Particle Swarm Optimization
RISC	Reduced Instruction Set Computer
RMI	Regional Mutual Information
RSSE	Root SSE
SIMD	Single Instruction, Multiple Data
SoC	System-on-Chip
SSE	Sum of Squared Errors
SRAM	Static RAM
ZNCC	Zero-Mean Normalized Cross-Correlation

References

1. Szeliski, R. *Computer Vision: Algorithms and Applications*; Springer: London, UK, 2010.
2. Maggio, E.; Cavallaro, A. *Video Tracking: Theory and Practice*; John Wiley & Sons: Hoboken, NJ, USA, 2011.
3. Kristan, M.; Leonardis, A.; Matas, J.; Felsberg, M.; Pflugfelder, R.; Kämäräinen, J.K.; Danelljan, M.; Zajc, L.Č.; Lukežič, A.; Drbohlav, O.; et al. The eighth visual object tracking VOT2020 challenge results. In *European Conference on Computer Vision*; Springer: Cham, Switzerland, 2020; pp. 547–601.
4. Forsyth, D.A.; Ponce, J. *Computer Vision: A Modern Approach*; Prentice Hall: New York, NY, USA, 2012.
5. Olague, G.; Hernández, D.E.; Llamas, P.; Clemente, E.; Briseño, J.L. Brain programming as a new strategy to create visual routines for object tracking. *Multimed. Tools Appl.* **2019**, *78*, 5881–5918. [CrossRef]
6. Hare, S.; Golodetz, S.; Saffari, A.; Vineet, V.; Cheng, M.M.; Hicks, S.L.; Torr, P.H. Struck: Structured output tracking with kernels. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *38*, 2096–2109. [CrossRef] [PubMed]
7. Asgarizadeh, M.; Pourghassem, H.; Shahgholian, G. Robust object tracking using regional mutual information and normalized cross correlation. In Proceedings of the 2012 Fourth International Conference on Computational Intelligence and Communication Networks, Mathura, India, 3–5 November 2012; pp. 411–415.
8. Liang, Z.; Liang, C.; Zhang, Y.; Mu, H.; Li, G. Tracking of Moving Target Based on SiamMask for Video SAR System. In Proceedings of the 2019 IEEE International Conference on Signal, Information and Data Processing (ICSIDP), Chongqing, China, 11–13 December 2019; pp. 1–4.
9. Hu, J.L.; Tang, X.W.; Qiu, J.N. Analysis of the influences of sampling bias and class imbalance on performances of probabilistic liquefaction models. *Int. J. Geomech.* **2017**, *17*, 04016134. [CrossRef]
10. Xiao, Y.; Pun, C.M.; Liu, B. Adversarial example generation with adaptive gradient search for single and ensemble deep neural network. *Inf. Sci.* **2020**, *528*, 147–167. [CrossRef]
11. Olague, G.; Hernandez, D.E.; Clemente, E.; Chan-Ley, M. Evolving head tracking routines with brain programming. *IEEE Access* **2018**, *6*, 26254–26270. [CrossRef]
12. Olague, G.; Ibarra-Vázquez, G.; Chan-Ley, M.; Puente, C.; Soubervielle-Montalvo, C.; Martinez, A. A deep genetic programming based methodology for art media classification robust to adversarial perturbations. In *International Symposium on Visual Computing*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 68–79.
13. Ibarra-Vazquez, G.; Olague, G.; Puente, C.; Chan-Ley, M.; Soubervielle-Montalvo, C. Automated design of accurate and robust image classifiers with brain programming. In Proceedings of the 2021 Genetic and Evolutionary Computation Conference Companion, Lille, France, 10–14 July 2021; pp. 1385–1393.
14. Ibarra-Vazquez, G.; Olague, G.; Chan-Ley, M.; Puente, C.; Soubervielle-Montalvo, C. Brain Programming is Immune to Adversarial Attacks: Towards Accurate and Robust Image Classification using Symbolic Learning. *arXiv* **2021**, arXiv:2103.01359.
15. Galoogahi, H.K.; Fagg, A.; Huang, C.; Ramanan, D.; Lucey, S. Need for speed: A benchmark for higher frame rate object tracking. In Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 1125–1134.
16. Morsi, N.N.; Abdelhalim, M.B.; Shehata, K.A. Efficient hardware implementation of PSO-based object tracking system. In Proceedings of the 2013 International Conference on Electronics, Computer and Computation (ICECCO), Ankara, Turkey, 7–8 November 2013; pp. 155–158.
17. Nogueira, B.; Barboza, E. A FPGA-based accelerated architecture for the Continuous GRASP. *Computing* **2020**, 103, 1–20. [CrossRef]
18. Prongnuch, S.; Wiangtong, T. Heterogeneous computing platform for data processing. In Proceedings of the 2016 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), Phuket, Thailand, 24–27 October 2016; pp. 1–4.
19. Liu, Y.; Zhu, H. A survey of the research on power management techniques for high-performance systems. *Softw. Pract. Exp.* **2010**, *40*, 943–964. [CrossRef]
20. Bean, A. Improving Memory Access Performance for Irregular Algorithms in Heterogeneous CPU/FPGA Systems. Ph.D. Thesis, Imperial College of Science, Technology and Medicine, London, UK, 2016.
21. Martin, G.; Chang, H. System-on-Chip design. In Proceedings of the 2001 4th International Conference on ASIC, Shanghai, China, 23–25 October 2001; pp. 12–17.
22. Kaeli, D.R.; Mistry, P.; Schaa, D.; Zhang, D.P. *Heterogeneous Computing with OpenCL 2.0*; Morgan Kaufmann: Waltham, MA, USA, 2015.
23. Wolf, W. A decade of hardware/software codesign. *Computer* **2003**, *36*, 38–43. [CrossRef]
24. Al-Zoubi, A.; Tatas, K.; Kyriacou, C. Towards Dynamic Multi-task Schedulling of OpenCL Programs on Emerging CPU-GPU-FPGA Heterogeneous Platforms: A Fuzzy Logic Approach. In Proceedings of the 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Nicosia, Cyprus, 10–13 December 2018; pp. 247–250.
25. Olague, G.; Puente, C. The honeybee search algorithm for three-dimensional reconstruction. In Proceedings of the Workshops on Applications of Evolutionary Computation, Budapest, Hungary, 10–12 April 2006; pp. 427–437.
26. Olague, G. *Evolutionary Computer Vision: The First Footprints*; Springer: Berlin/Heidelberg, Germany, 2016.
27. Tomassini, M. Evolutionary algorithms. In *Towards Evolvable Hardware*; Springer: Berlin/Heidelberg, Germany, 1996; pp. 19–47.
28. Blum, C.; Merkle, D. *Swarm Intelligence: Introduction and Applications*; Springer: Berlin/Heidelberg, Germany, 2008.

29. Clerc, M. *Particle Swarm Optimization*; Wiley: Hoboken, NJ, USA, 2010; Volume 93.
30. Karaboga, D.; Basturk, B. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. *J. Glob. Optim.* **2007**, *39*, 459–471. [CrossRef]
31. Dorigo, M.; Birattari, M.; Stutzle, T. Ant colony optimization. *IEEE Comput. Intell. Mag.* **2006**, *1*, 28–39. [CrossRef]
32. Orr, M.S.; Che, S.; Yilmazer, A.; Beckmann, B.M.; Hill, M.D.; Wood, D.A. Synchronization using remote-scope promotion. *ACM SIGARCH Comput. Archit. News* **2015**, *43*, 73–86. [CrossRef]
33. Tan, Y.; Ding, K. A survey on GPU-based implementation of swarm intelligence algorithms. *IEEE Trans. Cybern.* **2015**, *46*, 2028–2041. [CrossRef] [PubMed]
34. Perez-Cham, O.E.; Puente, C.; Soubervielle-Montalvo, C.; Olague, G.; Aguirre-Salado, C.A.; Nuñez-Varela, A.S. Parallelization of the honeybee search algorithm for object tracking. *Appl. Sci.* **2020**, *10*, 2122. [CrossRef]
35. ARM. *AMBA AXI and ACE Protocol Specification*; ARM: Cambridge, UK, 2011.
36. Churiwala, S.; Hyderabad, I. *Designing with Xilinx® FPGAs*; Springer: Cham, Switzerland, 2017.
37. Smeulders, A.W.; Chu, D.M.; Cucchiara, R.; Calderara, S.; Dehghan, A.; Shah, M. Visual tracking: An experimental survey. *IEEE Trans. Pattern Anal. Mach. Intell.* **2013**, *36*, 1442–1468.
38. Perez-Cham, O.E.; Puente, C.; Soubervielle-Montalvo, C.; Olague, G.; Castillo-Barrera, F.E.; Nunez-Varela, J.; Limon-Romero, J. Automata design for honeybee search algorithm and its applications to 3D scene reconstruction and video tracking. *Swarm Evol. Comput.* **2021**, *61*, 100817. [CrossRef]
39. Schaumont, P.R. *A Practical Introduction to Hardware/Software Codesign*; Springer: Boston, MA, USA, 2012.
40. Palermo, G.; Silvano, C.; Zaccaria, V. Discrete particle swarm optimization for multi-objective design space exploration. In Proceedings of the 2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools, Parma, Italy, 3–5 September 2008; pp. 641–644.
41. Tsai, C.C.; Huang, H.C.; Lin, S.C. FPGA-based parallel DNA algorithm for optimal configurations of an omnidirectional mobile service robot performing fire extinguishment. *IEEE Trans. Ind. Electron.* **2010**, *58*, 1016–1026. [CrossRef]
42. Rodriguez, A.; Moreno, F. Evolutionary computing and particle filtering: A hardware-based motion estimation system. *IEEE Trans. Comput.* **2015**, *64*, 3140–3152. [CrossRef]
43. Elkhani, N.; Muniyandi, R.C.; Zhang, G. Multi-objective binary PSO with kernel P system on GPU. *Int. J. Comput. Commun. Control.* **2018**, *13*, 323–336. [CrossRef]
44. Xilinx. *Zynq-7000 SoC Data Sheet: Overview*. 2018. v1.11.1. Available online: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf (accessed on 30 December 2021).
45. Kechiche, L. Hardware acceleration for deep learning of image classification. In Proceedings of the 2021 International Conference of Women in Data Science at Taif University (WiDSTaif), Taif, Saudi Arabia, 30–31 March 2021; pp. 1–5.
46. Rabiai, M.; Senouci, M.R.; Senouci, A.; Busawon, K.; Laurent, D. A hardware solution to overcome the bandwidth limitation of drone jamming platforms. In Proceedings of the 2020 12th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP), Porto, Portugal, 20–22 July 2020; pp. 1–4.
47. Qureshi, M.A.; Munir, A. PUF-IPA: A PUF-based identity preserving protocol for internet of things authentication. In Proceedings of the 2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 10–13 January 2020; pp. 1–7.
48. Xilinx. *7 Series FPGAs Data Sheet: Overview*. 2020. v2.6.1. Available online: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf (accessed on 30 December 2021).
49. Smith, D.J. *HDL Chip Design: A Practical Guide for Designing, Synthesizing and Simulating ASICs and FPGAs Using VHDL or Verilog*; Doone Publications: Madison, AL, USA, 1998.
50. Jahanirad, H.; Karam, H. BIST-based Testing and Diagnosis of LUTs in SRAM-based FPGAs. *Emerg. Sci. J.* **2017**, *1*, 216–225. [CrossRef]
51. Volder, J. The CORDIC computing technique. In Proceedings of the 1959 Western Joint Computer Conference, San Francisco, CA, USA, 3–5 March 1959; pp. 257–261.
52. Xilinx. *Divider Generator*. 2021. v5.1. Available online: https://www.xilinx.com/support/documentation/ip_documentation/div_gen/v5_1/pg151-div-gen.pdf (accessed on 30 December 2021).
53. Boumaza, A.M.; Louchet, J. Dynamic flies: Using real-time parisian evolution in robotics. In *Workshops on Applications of Evolutionary Computation*; Springer: Cham, Switzerland, 2001; pp. 288–297.
54. Bitam, S.; Batouche, M.; Talbi, E.G. A survey on bee colony algorithms. In Proceedings of the 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), Atlanta, GA, USA, 19–23 April 2010; pp. 1–8.
55. Deb, K. Multi-objective optimisation using evolutionary algorithms: An introduction. In *Multi-Objective Evolutionary Optimisation for Product Design and Manufacturing*; Springer: London, UK, 2011; pp. 3–34.
56. Crist, E. Can an insect speak? The case of the honeybee dance language. *Soc. Stud. Sci.* **2004**, *34*, 7–43. [CrossRef]
57. Deb, K.; Beyer, H.G. Self-adaptive genetic algorithms with simulated binary crossover. *Evol. Comput.* **2001**, *9*, 197–221. [CrossRef] [PubMed]
58. Goldberg, D.E.; Richardson, J. Genetic algorithms with sharing for multimodal function optimization. In Proceedings of the International Conference on Genetic Algorithms, Hillsdale, NJ, USA, October 1987; pp. 41–49.

59. Khare, V.; Yao, X.; Deb, K. Performance scaling of multi-objective evolutionary algorithms. In *International Conference on Evolutionary Multi-Criterion Optimization*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 376–390.
60. Bätz, M.; Richter, T.; Garbas, J.U.; Papst, A.; Seiler, J.; Kaup, A. High dynamic range video reconstruction from a stereo camera setup. *Signal Process. Image Commun.* **2014**, *29*, 191–202. [CrossRef]
61. Lin, C.; Li, Y.; Xu, G.; Cao, Y. Optimizing ZNCC calculation in binocular stereo matching. *Signal Process. Image Commun.* **2017**, *52*, 64–73. [CrossRef]
62. Collett, D. *Modelling Survival Data in Medical Research*; Chapman & Hall: London, UK, 1994.
63. Riesgo, T.; Torroja, Y.; De la Torre, E. Design methodologies based on hardware description languages. *IEEE Trans. Ind. Electron.* **1999**, *46*, 3–12. [CrossRef]
64. McFarland, M.C.; Kowalski, T.J. Incorporating bottom-up design into hardware synthesis. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **1990**, *9*, 938–950. [CrossRef]
65. Mano, M.M. *Computer Systems Architecture*; Prentice-Hall: Englewood Cliffs, NJ, USA, 1982.
66. Hopcroft, J.E.; Motwani, R.; Ullman, J.D. Introduction to automata theory, languages, and computation. *ACM SIGACT News* **2001**, *32*, 60–65. [CrossRef]
67. Zhang, D.; Zheng, Z. Joint Representation Learning with Deep Quadruplet Network for Real-Time Visual Tracking. In Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 19–24 July 2020; pp. 1–8.