FPGA-SoC Based Object Tracking Algorithms: A Literature Review

Marwan Abdulkhaleq Al-yoonus*
marwanathy1972@uomosul.edu.iq

Saad Ahmed Al-kazzaz**
kazzazs60@uomosul.edu.iq

* Electrical Engineering Departement, College of Engineering, University of Mosul, Mosul, Iraq
** Mechatronics Engineering Departement, College of Engineering, University of Mosul, Mosul, Iraq

Received: March 13th, 2023 Received in revised form: April 20th, 2023 Accepted: April 27th, 2023

ABSTRACT

Systems for object detection and tracking are becoming increasingly important in practical applications today. Many research and development groups are interested in improving the performance of such systems, and numerous methods have been developed and proposed. Additionally, computer vision is constantly developing and implemented on reconfigurable and embedded systems. The purpose of this study is to present past and recent research works in the field of visual tracking systems that used FPGA and FPGA-SoC platforms. The study includes a brief description of several popular algorithms related to the main characteristics and in which field is preferred. Resource utilization was also considered in this study to present the most and the least resources used to implement different algorithms. The study found that flip-flops (FF) and lookup tables (LUT) are usually used, while BRAM, DSP, and multipliers had the lowest percentage utilization. Due to the recent development in the production of advanced processing systems, there is an increase focusing on employing FPGA-SoC platforms in visual surveillance systems. The reason behind that is their ability to implement complex processing using both hardware and software co-design to gain high performance in less design time compared with using only FPGA-based platforms.

Keywords:

Background subtraction; detection algorithm; frame rate; tracking algorithm; FPGA resources.

==

1. INTRODUCTION

Moving object tracking is recently employed in wide applications for example; military, driverless vehicles, industrial, etc. The need for visual object tracking is due to the lack of another tracking system under different special cases. Electromagnetic waves and sonic waves are used for many years for detecting and tracking moving objects. Unfortunately, such systems are inefficient, especially for short distances and harsh environments in addition drone detection and tracking nowadays become more interesting for many researchers due to the failure of radar systems and other traditional tracking systems from this point of view, visual tracking become the trend of many research groups. In general visual tracking systems depend open the sensors (such as a digital camera) and hardware as well as software algorithms.

Object tracking is a central task contained by computer vision systems. The production of high-powered computers, the accessibility of high class and low-cost video cameras, and the growing need for robotic video analysis has created an excessive deal of attention in object tracking algorithms [1]. Tracking moving object, guessing the area of the moving object can really reduce extensive searching and make tracking system performance faster [2]. Practically, no idealistic system structure exists to process perfectly all kinds of problems within changed background models. To gain actual implementation for this kind of system, trade-offs mostly be made between system the robustness of the system and the system performance like resolution, frame rate, and so on. The main bottleneck of many image processing systems is the Memory usage [3].

Many different approaches have been proposed in the last 40 years, to determine the optical flow which is a technique used for determining the motion of objects in a video by measuring the change in pixel intensities over a set of frames [4]. An optical flow is incorporated into computer vision systems that carry out tasks like object detection and tracking detection [5].

The implementation of algorithms using FPGAs offer a good balance between development flexibility, algorithm testing and re-configurability, real-time performance, and costs [6] [7]. A SoC enables the integration of current hardware (HW) acceleration and software (SW) libraries into a single, small device. As a result, this technique enables reductions in both size and power usage [8] [9].

Typically, the algorithms are not ideal for hardware implementation as such, keep the amount of logic in minimum. Reducing the total number of element logics also present smaller power dissipation, which is significant matter in robotic and mobile applications. An outcome of adjusting algorithm to hardware implementation may result a reduction in accuracy. Accuracy reduction is not a problem in some applications because accuracy is not a big deal in that particular application. Many of the systems introduced in this review paper are suitable for adapting the algorithm to FPGA but others need to use both hardware/software implementations [1].

It is challenging to track an item, and it becomes even more challenging when it must be done on an embedded device with restricted resources. The first choice is to separate hardware and software, with software having the advantages of flexibility and speedier design and hardware having the advantage of high throughput [10]. Pre-filtering and feature detection are performed at the pixel level by the FPGA [11].

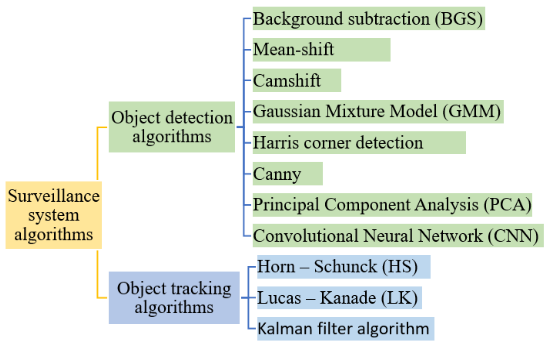
This paper is concerned on the research papers published from 2009 and upward. All the chosen papers (Nineteen) used an FPGA and FPGA-SoC platforms to process image or video signals. The selected papers subject depending on the implementation of object detection and object tracking algorithms on different devices. In most of the research papers, the review was focused on; the used algorithm, image size, frame rate (throughput), used FPGA device, resources utilization, and Power consumption.

This paper is organized in five sections. After the first section, introduction, an overview of commonly used algorithms in visual systems is presented in section 2. The literature survey was given in section 3. Important parameters summary gained from the survey section was outlined in section 4. Finally, the discussion and conclusions part found in section 5.

2. OBJECT DETECTION AND TRACKING ALGORITHMS

To recognize an object from a set of photos, a method or algorithm is needed to pick out key features [12]. Various algorithms have been put forth from time to time to increase the tracking process' efficiency. However, no algorithm has yet been created that will function well in all environmental situations [13]. Some algorithms may not function properly when the camera is used to record the video moves, while others may not function properly in conditions of intense lighting. Due to the loss of some information when converting a 3-D environment to a 2-D image, real-time processing, noise in the images, changes in scene illumination, complex object shapes, motion, and partial or complete occlusion, the tracking process is in and of itself an extremely complex task [14] [15].

There are many algorithms proposed in the field of visual object tracking. Some of them used as preprocessing stage in visual tracking system which include detecting the objects and others used for tracking objects [16]. Figure 1 summarized the most common algorithms used in visual detecting and tracking systems selected algorithms.
Visual detection and tracking algorithms can be split into two parts; detection algorithms and tracking algorithms. A short description of the commonly used algorithms are presented in the next subsection.



Surveillance system algorithms
Object detection algorithms
Background subtraction (BGS)
Mean-shift
Camshift
Gaussian Mixture Model (GMM)
Harris corner detection
Canny
Principal Component Analysis (PCA)
Convolutional Neural Network (CNN)
Object tracking algorithms
Horn – Schunck (HS)
Lucas – Kanade (LK)
Kalman filter algorithm

Fig.1 Common detection and tracking algorithms.

2.1 Object detection algorithms

The first step or the preprocessing stage in surveillance system is object detection. A short description of the common detection algorithms shown in figure 1are presented here:

a- Back Ground Subtraction (BGS)

Detects the actual background and extracts objects that do not belong to it. To recognize between moving and stationary objects, the BGS algorithm requires three sequential frames and a reference image of a stationary background [17] [18]. In order to perform automatic threshold selection, subtraction operation, and pixel-wise classification, a background subtraction model based on Horprasert acquires a reference image to model the background of the scene. This algorithm work correctly when the camera is fixed [15] [19].

b- Mean-Shift algorithm

Due to the unsupervised nature of this algorithm, it can be used in a variety of autonomous applications where no input parameters are provided by the user. The main challenge is computational complexity and scales poorly with both the number of pixels N^2 and number of iterations (k) as $O(kN^2)$ [20].
Implementing parallel processing and pipelining of such pixels on the FPGA resulting in a decrease in computational complexity for real-time application system. In contrast to its effectiveness when the camera is moving, it is a requirement of this algorithm that there be no occlusion [15].

c- The Camshift algorithm

This is based on the Mean-shift algorithm and it was improved to use the object color information to inform the Mean-Shift algorithm [21]. This method is unaffected by changes in object shape. It can effectively address the partial occlusion and object deformation problems with a higher operating efficiency. The limitation is that the histogram of the target image records the probability of the color appearing, so the algorithm requires that the object be manually specified before it can start [2].

d- Gaussian mixture model GMM algorithm

This type of algorithm is considered a probabilistic algorithm that is especially well suited to detecting moving objects in multimodal backgrounds with repetitive motion-showing objects like waves, moving leaves, and flickering light. In the presence of changes in illumination, the GMM algorithm performs well [22] [23].

e- Harris corner (HC) detection algorithm

It has enormous parallel processing power, pixel-wise operation, and operator noise immunity. It performs well when detecting L-shaped corners [11].

f- Canny algorithm

It is an optimal edge detection technique which provide good detection, clear response, and good localization. It is suitable for implementation in a pipeline parallel architecture on the FPGA [24] [25].

g- Principal Component Analysis (PCA) algorithm

It is a method used in many different disciplines, such as artificial vision, power electronics, and statistics. The PCA technique for image processing enables the reduction of redundant information (retaining only essential information) of the initial variables and the evaluation of the degree of similarity between two or more images by analyzing only the fundamental features present in the transformed space [26].

h- Convolutional Neural Network (CNN) algorithm

A CNN is a type of multi-layered neural network. In computer vision, CNN has been rising to greater prominence. The major benefit of CNNs is their capability of self-learning, meaning that the more images they are exposed to, the better they become at classifying objects [12] [27]. CNN requires a lot of computing power. During deployment and training, it uses a tremendous amount of computing power. Due to the potential tradeoff between power consumption and reconfigurability, FPGA-based CNN accelerators received a lot of research attention [28].

2.2 Object tracking algorithms

The motion can be determined in a video sequence by subtracting two frames acquired after each other. This allows one to see where the movement (change) has occurred but does not provide much information regarding its direction or speed. However, more sophisticated vision systems require this knowledge. Consequently, the idea of optical flow was presented [29] [5]. Optical flow is a vector that represents the motion of a target or an object in an image sequence (video) [30]. There are various methods for optical flow computation:

A)- Horn – Schunck (HS) algorithm

Horn–Schunck is a global regularized algorithm that obtains a globally optimized solution with iterative calculation, so the characteristics of the entire image are considered, in other words, on the whole picture, the optical flow should be uniformly smooth. As a result, each pixel's calculated flow in a small neighborhood is similar [5] [30].

B)- Lucas – Kanade (LK) algorithm

Also known gradient-based optical flow estimation algorithm. The LK algorithm took into account a small neighborhood of each pixel rather than looking for the global minimum features of the entire image. Its theory is based on the observation that a pixel moves in the same way as its nearest neighbors.

As a result, unlike the HS algorithm, the assumption introduced by the LK algorithm only needs to be satisfied locally [5]. Embedded hardware can successfully implement the Lucas – Kanade algorithm [11]. In the case of significant motion between consecutive frames, this algorithm's accuracy declines. Use of a high-frame-rate scheme is the solution [31] [32] [33].

C)- Kalman filter algorithm

When measurement values are uncertain, the Kalman filter, a minimum mean-square error estimator, provides the most accurate estimation of a linear dynamic system model, including an object's position, velocity, and true measurements [34]. Object tracking and motion detection in dynamically positioned vehicles are two typical applications of Kalman filter adaptation [35].

3. LITERATURE SURVEY

In this section, the research papers which were highlighted in this review paper are separated into two groups, the first group implemented their proposed systems using only FPGA and the other group implemented their proposed systems by using FPGA-SoC platforms.

3.1 Research work based on FPGA platforms

Hongtu Jiang et al. [3] proposed a dedicated hardware architecture for real-time segmentation at VGA resolution and 25 frames per second. The authors presented an FPGA platform with a number of memory access reduction schemes, which reduces memory bandwidth by more than 70%. The video sequences with three Gaussian distributions per pixel were used to achieve the real-time segmentation performance. The off-chip DDR SDRAM that houses the Gaussian parameters. Hardware complexity was reduced by updating only one Gaussian parameter at a time. The proposed hardware's bottleneck is memory usage.

A complete implementation of the PCA algorithm was presented by I. Bravo et al. [26] on reconfigurable hardware (FPGA) devices to detect new objects in a scene. Different components of the PCA algorithm's traditional sequential execution have been parallelized. FPGA was used to implement the entire system. Each algorithm part's computation time is also declared. A 128 MB SDRAM memory bank was included in the system and was external to the FPGA.

The lane detection and tracking procedures were created and implemented by Marzotto et al. [7] in a single FPGA device. The suggested system architecture consists of self-contained logic modules that don't require the assistance of programmable microcontrollers, DSP processors, or external memories because every module was fully implemented inside the FPGA. The authors clearly described the steps in the pre-processing pipeline procedure. The suggested system is flexible enough to adapt to different road conditions without pre-setting. The tracking algorithm is made up of three separate Kalman filters (KF) that each work on three different parameters. The Xilinx System GeneratorTM for DSP is the foundation for the entire FPGA system implementation. Due to the system functionality only utilizing about 30% of the Spartan-3A's hardware resources, an FPGA with fewer hardware resources can be used.

F. Barranco et al. in [31] implemented the optical flow core and the multi-scale extension using high level Handel-C. The PCI interfaces, off-chip memory, and memory controller unit (MCU) are all implemented using the RTL language VHDL. The system architecture, the pipelined stage scheme, and the primary hardware resources were all described by the authors. Additionally, demonstrated that the highest clock operation frequency was possible with minimal resource use. There were two different abstraction levels used in the hardware implementation. In the used device, the mono-scale implementation employed 10%–15% of the resources, whereas the multi-scale used about 60%. While the frame rate was decreased to about a tenth of the mono-scale approach. The authors increased the precision results by about three times.

I. Ishii et al. used an enhanced gradient-based algorithm based on the LK method in Ref [32], which can adaptively choose an artificially variable frame rate in accordance with the estimated optical flow's (OF's) amplitude to precisely detect it for entities moving at both high and low speeds in the same image's grayscale 10-bit per pixel representation. The optical flows were estimated at 1000 f/s for every of the 1024×1024-pixel image's 1024 block regions of 32×32 pixels. Software on the PC carried out the operations in the computation of the 1024 blocks at the block level.

Authors employ two FPGAs: one is used for processing and displaying images, and the other for implementing user algorithms on hardware.

M. Genovese et al. in Ref.[36] proposed an FPGA implementation of Open source Computer Vision software library which is developed by Intel (OpenCV). Variety platforms used to synthesiz and implemente GMM algorithm, including the Stratix-IV Altera FPGA and the Virtex-6, Virtex-5, and Spartan-3 Xilinx FPGA. The circuit used three Gaussian distributions for each pixel when processing grayscale videos. The proposed circuit processes 13 parameters for each pixel, including the luminance value and 12 Gaussian parameters for the pixel's statistical model. Utilizing the Stratix-IV platform from Altera FPGA, high frame rate and operating frequency were achieved.

A pipelined, parallel optical flow algorithm developed by G. K. Gultekin et al. [37] significantly boosts system throughput by using multiple clock domains. The memory interface circuit operates at a higher clock rate than the calculation modules, which aids in removing the design's memory bottleneck. Authors used the divide by powers of two methods to approximate the division operation. The performance of the proposed hardware implementation's algorithm is compared to that of a PC implementation by the authors. The comparison revealed that the reference FPGA implementation ran faster than the PC implementation by about 146 times. Additionally, the power used by FPGA implementation is only 844.38mW, or about 1/40 of the power used by a 1.66 GHz personal computer processor. The 200 MHz (SSRAM controller) and 50 MHz (optical flow controller) clocks are produced by a phase locked loop (PLL) internal circuit.

In Ref. [11], M. Tomasi et al. proposed an FPGA+DSP system that outperformed ARM+DSP and DSP only configurations by about 20 and 3 times, respectively. In the FPGA platform, a fine-grain pipeline was utilized by the authors to implement Harris corner detection, with a data rate of 60 megabytes per second (MBPS). In order to achieve a total frame rate of 160 fps for VGA images, the DSP simultaneously receives and tracks the features that the FPGA has detected. For comparison, the performance was split into detection and tracking. An IP core for Harris corner detection was created. The FPGA-based detection algorithm was executed, and the DSP board—which serves as the FPGA's co-processor—ran the LK algorithm to track the detected feature points. Authors didn't state the hardware architecture design of the used algorithms. Instead, they presented an analysis of the processing times and hardware performances in three architectures used: FPGA, DSP, and ARM. FPGA proved the faster speed performance of 4.9ms for VGA images compared with 138ms for ARM and 10.6ms for DSP.

In their novel hardware architecture, and instead of storing the original input image, H.-S. Seong et al.[33] suggested storing the input image following the Gaussian filtering process. To reduce the external memory access to a quarter of the original data, the Gaussian-filtered image was downsampled in both the horizontal and vertical directions. The total memory access is reduced by 75% when using this technique (2:1 sub-sampling) in both the vertical and horizontal directions. With a slight increase in hardware resources, the external bandwidth was reduced. Instead of using multipliers, which consume more time and resources, the authors used the streamlined Gaussian coefficient, which only requires shifter and adder operations. The authors went into great detail about the suggested hardware organization for the LK algorithm design.

S. Sajjanar et al. in Ref [15] presented a complete system module including various sub-modules which are the controller, storage, display, and camera capture modules. The description of the RTL blocks ports of each Module explained clearly. The system used three memory modules: a display VGA module, a frame buffer module, and a background memory module. The incoming and reference frames were stored in the first two, and the resultant frame was stored in the third. The camera register was configured to automatically obtain data in the YUV form, which represent each pixel of an image using 24 bits (eight bits for each Y, U, and V). In later stages, only the 8 bits corresponding to Y, which denotes an image's grayscale, are used.

A. Arif et al. [17] described an implementation of an algorithm to process traffic camera image sequences in real-time. The algorithm requires four frames (images) as input: the frame being studied, the frame before it, the frame after it, and the reference stationary background. The algorithm compares the corresponding pixels from three subsequent frames to determine the weighted difference. When the difference is 0, it means that the corresponding pixel hasn't moved at all. Thus, there is no need to update the reference background. The algorithms are written in OpenCL code. The power consumption of the background subtraction and Lucas-Kanade algorithms on each platform, FPGA, CPU, and GPU was compared by the authors. FPGAs' computational prowess and power effectiveness demonstrated that they are excellent candidates for applications that call for intensive data processing, particularly in real-time.

P. Hobden et al. in Ref.[12] provide a solution method to overcome the limited floating-point resources but keep running in real-time operation. Two modules were included in the proposed method: tracking and detection of unmanned aerial vehicles (UAVs) using neural networks (NNs). The tracking module used a background-differencing algorithm, while the UAV detection used a modified CNN algorithm. Authors compared the implementation of MATLAB and Xilinx Deep Learning Processor Unit (DPU) on the UltraScale ZCU102 against their model by using the data set of the same images.

3.2 Research work based on FPGA-SoC platforms

U. Ali and M. B. Malik [21] presented a hardware/software co-design architecture for the well-known kernel-based mean shift tracking algorithm. The target's color histogram was used in the design as a tracking feature. The target was located in the following images by maximizing the statistical match of the color distributions. The system was able to track multiple targets at frame rates of up to hundreds of frames per second by localizing them using gradient-based iterative search as opposed to exhaustive search. The authors discussed how long various tasks took to complete. All computationally intensive tasks were mapped on hardware to achieve maximum frame rate, while the prediction filter and main tracking loop update were implemented in software to simplify target initialization procedures by taking inputs from the user.

An FPGA-based embedded architecture with low degradation that can extract the background in environments with limited resources was proposed by R. Rodriguez-Gomez et al. [19]. The MicroBlaze processor, which was employed to create the benchmark background model and to update it over time, is a part of the suggested architecture. Using fixed-point operations, a specific hardware module carried out the subsequent stages of subtraction and pixel-by-pixel classification. The authors ran the system modules with different clock frequency domains. To make the FPGA clock supply networks simpler, the authors ran the suggested IP core BGS at the same frequency as MicroBlaze and system buses. The architecture's hardware complexity was greatly reduced during the training phase by precalculating and storing a number of constants. Division operations are avoided by using multiplications instead, which use less hardware. The fundamental Horprasert model was enhanced to effectively handle shadows, which represents a significant advancement in common scenes.

S. Guo et al. in Ref. [38] used the processing (PS) to run a Gaussian background model-based detection program to detect moving objects entering the field of view. The reconfigurable area of the Programmable Logic (PL) was programmed with the accelerator portion of the Gaussian background model (FPGA subsystem). The Compressive tracking (CT) algorithm was employed for object tracking. After being converted to 320x240 8-bit grey scale, the acquired images are saved in the image preprocess module using DDR3 RAM. The diagrams for the background model and the tracking model were thoroughly explained by the authors. Additionally, a performance comparator with and without a hardware accelerator was presented. The average tracking frame rate for the proposed system was 9.48 times faster than that of the ARM processor-based pure software solution. There were some unavoidable issues with the proposed system. Only videos with a static background are processed by the detection component. Additionally, the drift issue restricted the tracking component, which ultimately caused the tracking operation to fail.

Because of the stability of the Mixture of Gaussian (MoG) algorithm for background subtraction, which is implemented in FPGA, G. Conti et al. in [8] selected it for HW acceleration. The robustness of the Kalman filter, which is based on a statistical model, led to its implementation in PCs for tracking. Gray scale and RGB were used by authors to measure the speed differences between them. Although the results from the RGB version of the algorithm were more accurate, it was less efficient than the gray-scale version. More pixels are simultaneously stored in FPGA memory using grayscale. The RGB version offers the ideal balance between accuracy and processing time.

J. G. Pandey used the mean shift-based moving object tracking algorithm in [39]. A detailed explanation of the algorithm implementation in the FPGA. the circuits utilized as intellectual property (IP) cores in the framework for the implementation of a mean shift algorithm based on the kernel for tracking moving objects. The computation of the Bhattacharyya coefficient, mean shift vector, and associated circuits implemented using fixed-point binary logarithmic and antilogarithmic units. The tracking algorithm was initially developed in C, tested using a number of saved video files, and then implemented using RTL-level VHDL code.

For the tracking and detection of multiple objects, P. Babu et al. [35] presented a multi-dimensional Kalman filter (MDKF) for linear systems with updated state vector and covariance equations. Additionally, the hardware multi-dimensional Kalman filter implementation on the Zynq SoC with efficient resource utilization was shown. On various benchmark datasets, the MDKF tracking algorithm was used to estimate performance and accuracy. The only resource limitation was the usage of DSP blocks in the Zynq SoC, which may run out as the number of states for measuring uncertainties increased.

To process different numbers of pixels concurrently depending on the scale and without using additional external memory to store temporal values, a multi-scale method, gradient-based algorithms, Lucas-Kanade and Horn-Schunck were implemented on a ZCU platform with a Zynq UltraScale+ MPSoC FPGA by K. Blachut et al.[5]. For each potential input vector, Look-Up Tables (LUT) were created with pre-calculated values. The four pixels per clock data format used in the 4K video stream allows authors to lower the lower frequency to 150 MHz needed for real-time processing.

P. Hobden et al. discovered a way to deal with the issue of scarce floating-point resources while preserving real-time applications in ref. [12]. The solution consists of modules for neural network-based UAV detection and tracking of unmanned aerial vehicles (UAVs). While the UAV detection used a modified CNN algorithm, the tracking module used a background-differencing algorithm. The CNN algorithm was used to deliver a feedback path so that it could be verified that the tracking had locked onto the right object and not the wrong one. In order to best allocate hardware resources on the PL unit, the authors implemented a few layers in the Advanced RISC Machines (ARM) PS core of the Zynq, and they also carried out the training using MATLAB on a PC.

4. SUMMARY OF THE REVIEWED PAPERS

In this section, the most crucial factors, including the algorithm chosen, the platform is chosen, the frame rate, the image resolution, the frequency of use, and the power consumption of the preceding research papers are collected in table 1 to simplify the performance comparison for the readers while table 2 shows the number of resources utilization and the percentage of the used resources to the total number for most of the selected papers that write the results in details. It is clear that multipliers are the less used resources compared with others because they drain FPGA resources.

From table 1, it is clear that there is a tradeoff between image (target) capture size and the frame rate. Other indication that can be obtained from the comparison, authors in ref. [33] [36] implement the same algorithm in different platforms while authors in ref. [5] implement two different algorithms on the same device as explained in table 2. From all the selected papers, most hardware implementations of detecting and tracking algorithms based on FPGA devices, one of the selected paper employed FPGA–DSP platforms and the remainder employed the popular recent technology based on hardware/software (co-design).

The extracted information from the mentioned researches about the numbers and resources percentage utilization related to; slices, FF, BRAM, DSP, LUT, and multipliers are collected in table 2. Although the algorithms and applications presented in this paper are different, a computation of the total resources for all the used devices in the research papers was presented in figure 2. The percent ratio between each resource type to the total resources (regardless of the type of resource) was calculated and plotted to give an overview of the most used resources among the others in such application to provide an indication for researchers and industrials. Any not specified resources by the researchers are not considered and not included in figure 2 so to reach a reasonable approximation ratio as possible.
Another comparison presented in figure 3 explains the percentage ratio between the usage of each resource type to the total number of the same resource.

From figure 2 and 3 it is shown that BRAM, DSP, and multipliers are the limited FPGA resources. It is clear that multipliers are the lower used compared with others because of their resource-consuming. Flip-flops (FF) are the most used resources and the most fabricated numbers among the others. Slices come in the second order after FF.

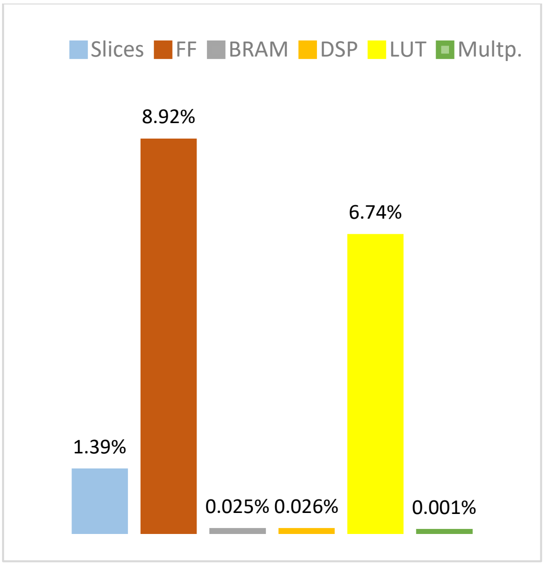


Fig. 2: The percentage ratio for each resource utilization to the sum of the total number of all resources type.

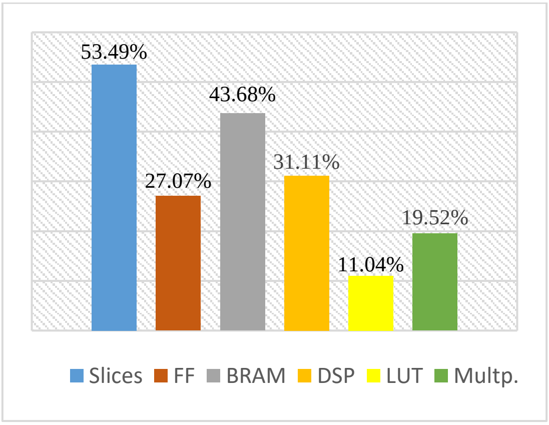


Fig 3: The normalized percent ratio for each resource utilization.

5. DISCUSSION AND CONCLUSIONS

In this review paper and from the selected subject area in the field of common detection and tracking algorithms in surveillance systems, some of the reviewed documents proposed the hardware implementation of one or two algorithms using just FPGA platform, other articles used both hardware/software Co-design (also called hybrid system).

From figure 2 it is clear that flip-flop (FF) is the most used among other FPGA resources. The other logic cell is the look-up table (LUT). On the other hand, multipliers are rarely used because it consumes more devices and computational time.

Table 1: Performance comparison of FPGA based for different visual algorithms of recent and earlier literatures. (-) indicates not specified value.

Ref.	**Algorithm**	**Platform**	**Frame rate/ Resolution**	***Freq.***	**Power consumption**	***year***
[3]	Segmentation algorithm	Virtex-II pro Vp30	640x480 25 fps	25MHz	_	2009
[21]	Mean shift tracking algorithm	Spartan-3e XC3S1600E	64x64 290 fps	50 MHz	_	2010
[26]	Principal Component Analysis (PCA)	XC2VP7	256 ×256 120 fps	-100 MHz -Sensor Clk.= 66MHz	_	2010
[7]	Lane Marking Pattern Search	Spartan-3A DSP 3400	752 x 480 30 fps	128.2 MHz	_	2010
[19]	BGS based on Horprasert	XC3SD3400 Spartan-3A	32.8 fps 1,024×1,024	66.5 MHz	5.76W	2012
[31]	Lucas-Kanade -*mono-scale -**multi-scale	Virtex-4 XC4vfx100	640 x 480 - *270 fps - **31.9 fps	- *83MHz -** 44MHz	_	2012
[32]	Lucas–Kanade (gray-level)	2-FPGA Xilinx XC2VP100	1000 fps 1024×1024	90MHz	_	2012
[40]	GMM (gray-scale)	Stratix-IV SGX230HF35C2	1920 × 1080 47 fps	98.12MHz	_	2013
[37]	Optical flow (based on HS)	EP2C70 Cyclone-2	256 x 256 257 fps	50 MHz	844.38mW	2013
[11]	-*Harris corner (detection) -**LK (tracking)	xcv4fx60 Virtex-4	-*29 fps 1,920x1,080 -**160 fps 640x480	62 MHz	-686mW, for FPGA -1,697mW, for DSP	2014
[33]	Lucas-Kanade (LK)	-Virtex-6	170 fps	94MHz	_	2015

		-Virtex-4	800x600			
[15]	Modified BG Subtraction	Zynq-7000 XC7Z020	30 fps 640x480	25MHz	_	2016
[38]	-Gaussian BG -CT (gray-scale)	Zynq-7000 XC7Z020	89.2 fps 320x240	667 MHz	2.99 W	2017
[17]	- BG subtraction - LK algorithm	Virtex-7 XC7VX690T-2	29 fps 1280 × 4	200 MHz	-2.760W (BG) -8.385 W (LK)	2018
[8]	Gaussian (MoG)	Zynq-7000 XC7Z020	640x480 162 fps	_	- 2.297W-RGB -2.025W-Gray	2020
[39]	Mean shift	Virtex-5 xc5vfx70t	60 fps 640 × 480	25.175 MHz	315 mW	2021
[35]	Kalman filter	ZynqTM-7000 Artix-7	91 fps -	140 MHz	780 mW	2021
[5]	- LK - HS	UltraScale + ZCU 104	60 fps 3840 × 2160	-300MHz(LK) -150MHz(HS)	5.70 W	2022
[12]	- BG -CNN	Zynq UltraScale XCZU9EG	54.67 fps -	220MHz	5.5W	2022

Table 2: Resource usage of the overall design on the FPGA devices.

Ref.	Slices	FF	BRAM	DSP	LUT	Multp.	**FPGA device**	**Platform**
[3]	6107 44.6%	4273 -	84 3.43%	-	-	-	VirtexII pro Vp30	FPGA
[21]	3160 20%	4187 14%	9 25%	-	3516 11%	3 8%	Spartan-3e XC3S1600E	Co-design H/S
[26]	4225 86%	-	40 91%	-	-	43 98%	XC2VP7	FPGA
[7]	8398 35.2%	-	32 25.4%	34 27%	-	-	Spartan-3A DSP 3400	FPGA
[19]	15522 65%	183689 38%	44 35%	48 38%	19616 41%	-	XC3SD3400 Spartan-3A	Co-design H/S
[31]	26036 61%	24694 29%	112 29%	62 38%	31796 37%	-	Virtex-4 XC4vfx100	FPGA
[32]	14312 32%	26551 30%	28 6%	-	4935 5%	80 18%	Xilinx XC2VP100	FPGA
[40]	1010 13.15%	0	-	0	1797 11.69%	0	Spartan3 (xc3s1000)	FPGA
	301 4.18%	0	-	0	844 2.93%	0	Virtex5 (xc5vlx50)	
	265 2.27%	0	-	0	922 1.98%	0	Virtex6 (xc6vlx75t)	
	81 0.88%	0	-	0	1150 0.63%	0	Stratix-IV SGX230HF3	
[37]	8086 11.9%	151772 13.2%	-	-	-	6 4%	EP2C70 Cyclone-2	FPGA
[11]	5766 22%	-	59 25%	9 7%	5893 11%	-	xcv4fx60 Virtex-4	FPGA+ DSP
[33]	-	16219 2.13%	120 8.3%	54 6.25%	31305 -	-	Virtex-6 LX760	FPGA
	-	17139 9.5%	132 39%	54 56.25%	43228 24.37%	-	Virtex-4LX200	
[15]	125 1%	286 1%	120 85%	-	53200 1%	-	Zynq-7000 XC7Z020	FPGA
[38]	_	31907 30%	287 102.5%	173 78.6%	41163 77.4%	_	Zynq-7000 XC7Z020	Co-design H/S
[17]	-	241593 28%	1140 39%	1065 30%	264366 61%	-	Virtex 7 XC7VX690	FPGA
[8]	11,589 66.6%	11,588 10.89%	135 96.4 %	39 17.7 %	24,170 45.4%	-	Zynq-7000 XC7Z020	Co-design H/S
[39]	-	113 0.3%	20 13.5%	48 37.5%	1998 4.4%	-	Virtex-5 XC5VFX70	Co-design H/S
[35]	69997.5 82.35%	80885.28 76.02%	22.11 61.43%	199.98 90.9%	44299.64 83.27%	-	ZynqTM- Artix-7	Co-design H/S
-LK [5]	-	183688 40%	311 100%	861 50%	122734 53%	-	Xilinx UltraScale + ZCU 104	Co-design H/S
-HS	-	145872 32%	312 100%	523 30%	104728 45%	-		
[12]	-	-	223 24.45%	220 8.73%	48500 17.7%	-	Zynq XCZU9EG	Co-design H/S

Where; FF: Flip-Flops, BRAM: Block RAM, DSP: Digital Signal Processing, LUT: Look-up-table, Multp.: Multiplier.

By using gray-scale, more pixels can be stored in FPGA memory when memory usage is limited also the processing speed can be increased to obtain high frame rate.

Although using RGB version reduce the frame rate compared with using gray scale but using it get more accurate results for detection and tracking. RGB version is the solution for complex environments.

By using LUT for the constrained range, hardware resources can be used more sparingly and redundant data storage can be avoided. Also we can get highest clock operation frequency when use less resources.

When an algorithm has a computationally intensive task, it must be implemented in hardware to achieve the highest throughput (frame rate). However, if the algorithm needs to interface with the user, it is worthwhile to implement this portion of the algorithm in software because it will facilitate target initialization procedures by accepting user input. FPGAs' computational prowess and power effectiveness demonstrated that they are excellent candidates for applications that call for intensive data processing, particularly in real time.

Other researchers calculate some parameters that consume many resources and use lookup tables to save and then use them during operation.

In comparison to more conventional approaches, the CNN algorithm is an appropriate choice for devices with limited DSP slices which is used for floating-point implementation.

In some applications it is possible to simplify the hardware implementation so that shifter and adder operations can be used instead of multiplier that cost more resources and time.

The co-design approach, which was used to reduce the amount of hardware needed. Additionally, by optimizing the code at key points and interfaces, high-level languages like ImpulseC and RTL descriptions defined using VHDL enables a reduction in the implementation strategy and the achievement of high performance.

REFERENCES

[1] Sirpa Korhonen, "'Hardware Accelerated Visual Tracking Algorithms. A Systematic Literature Review," Aug. 2015. [Online]. Available: https://www.researchgate.net/publication/281088238

[2] S. Gong, C. Liu, Y. Ji, B. Zhong, Y. Li, and H. Dong, Advanced Image and Video Processing Using MATLAB, vol. 12. in Modeling and Optimization in Science and Technologies, vol. 12. Cham: Springer International Publishing, 2019. doi: 10.1007/978-3-319-77223-3.

[3] Hongtu Jiang, H. Ardo, and V. Owall, "A Hardware Architecture for Real-Time Video Segmentation Utilizing Memory Reduction Techniques," IEEE Trans. Circuits Syst. Video Technol., vol. 19, no. 2, pp. 226–236, Feb. 2009, doi: 10.1109/TCSVT.2008.2009244.

[4] R. Parekh, Fundamentals of image, audio, and video processing using MATLAB: with applications to pattern recognition, First edition. Boca Raton London New York: CRC Press, 2021.

[5] K. Blachut and T. Kryjak, "Real-Time Efficient FPGA Implementation of the Multi-Scale Lucas-Kanade and Horn-Schunck Optical Flow Algorithms for a 4K Video Stream," Sensors, vol. 22, no. 13, p. 5017, Jul. 2022, doi: 10.3390/s22135017.

[6] B. M. K. Younis, B. Sh. Mahmood, and F. H. Ali, "Reconfigurable Self-Organizing Neural Network Design and it's FPGA Implementation," (AREJ), vol. 17, no. 3, pp. 99–115, Jun. 2009, doi: 10.33899/rengj.2009.42925.

[7] R. Marzotto, P. Zoratti, D. Bagni, A. Colombari, and V. Murino, "A real-time versatile roadway path extraction and tracking on an FPGA platform," Computer Vision and Image Understanding, vol. 114, no. 11, pp. 1164–1179, Nov. 2010, doi: 10.1016/j.cviu.2010.03.015.

[8] G. Conti, M. Quintana, P. Malagón, and D. Jiménez, "An FPGA Based Tracking Implementation for Parkinson's Patients," Sensors, vol. 20, no. 11, p. 3189, Jun. 2020, doi: 10.3390/s20113189.

[9] M. Amiri, F. M. Siddiqui, C. Kelly, R. Woods, K. Rafferty, and B. Bardak, "FPGA-Based Soft-Core Processors for Image Processing Applications," J Sign Process Syst, vol. 87, no. 1, pp. 139–156, Apr. 2017, doi: 10.1007/s11265-016-1185-7.

[10] F. Kristensen, H. Hedberg, H. Jiang, P. Nilsson, and V. Öwall, "An Embedded Real-Time Surveillance System: Implementation and Evaluation," J Sign Process Syst Sign Image Video Technol, vol. 52, no. 1, pp. 75–94, Jul. 2008, doi: 10.1007/s11265-007-0100-7.

[11] M. Tomasi, S. Pundlik, and G. Luo, "FPGA–DSP co-processing for feature tracking in smart video sensors," J Real-Time Image Proc, vol. 11, no. 4, pp. 751–767, Apr. 2016, doi: 10.1007/s11554-014-0413-2.

[12] P. Hobden, S. Srivastava, and E. Nurellari, "FPGA-Based CNN for Real-Time UAV Tracking and Detection," Front. Space Technol., vol. 3, p. 878010, May 2022, doi: 10.3389/frspt.2022.878010.

[13] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," ACM Comput. Surv., vol. 38, no. 4, p. 13, Dec. 2006, doi: 10.1145/1177352.1177355.

[14] Dr. F. Ali, "Transformation Matrix for 3D computer Graphics Based on FPGA(English)," (AREJ), vol. 20, no. 5, pp. 1–15, Oct. 2012, doi: 10.33899/rengj.2012.61024.

[15] S. Sajjanar, S. K. Mankani, P. R. Dongrekar, N. S. Kumar, Mohana, and H. V. Ravish Aradhya, "Implementation of real time moving object detection and tracking on FPGA for video surveillance applications," in 2016 IEEE Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), Mangalore: IEEE, Aug. 2016, pp. 289–295. doi: 10.1109/DISCOVER.2016.7806248.

[16] Dr. S. A. Dawwd and U. T. Salim, "Systolic Video Stream Object Detector Using FPGA-E," (AREJ), vol. 22, no. 4, pp. 33–43, Sep. 2014, doi: 10.33899/rengj.2014.89977.

[17] A. Arif et al., "Performance and energy-efficient implementation of a smart city application on FPGAs," J Real-Time Image Proc, vol. 17, no. 3, pp. 729–743, Jun. 2020, doi: 10.1007/s11554-018-0792-x.

[18] N. A. Mandellos, I. Keramitsoglou, and C. T. Kiranoudis, "A background subtraction algorithm for detecting and tracking vehicles," Expert Systems with Applications, vol. 38, no. 3, pp. 1619–1631, Mar. 2011, doi: 10.1016/j.eswa.2010.07.083.

[19] R. Rodriguez-Gomez, E. J. Fernandez-Sanchez, J. Diaz, and E. Ros, "FPGA Implementation for Real-Time Background Subtraction Based on Horprasert Model," Sensors, vol. 12, no. 1, pp. 585–611, Jan. 2012, doi: 10.3390/s120100585.

[20] D. B. K. Trieu and T. Maruyama, "Real-time color image segmentation based on mean shift algorithm using an FPGA," J Real-Time Image Proc, vol. 10, no. 2, pp. 345–356, Jun. 2015, doi: 10.1007/s11554-012-0319-9.

[21] U. Ali and M. B. Malik, "Hardware/software co-design of a real-time kernel based tracking system," Journal of Systems Architecture, vol. 56, no. 8, pp. 317–326, Aug. 2010, doi: 10.1016/j.sysarc.2010.04.008.

[22] M. Genovese and E. Napoli, "FPGA-based architecture for real time segmentation and denoising of HD video," J Real-Time Image Proc, vol. 8, no. 4, pp. 389–401, Dec. 2013, doi: 10.1007/s11554-011-0238-1.

[23] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149), Fort Collins, CO, USA: IEEE Comput. Soc, 1999, pp. 246–252. doi: 10.1109/CVPR.1999.784637.

[24] Artur Zawadzki and Marek Gorgon, "Automatically controlled pan–tilt smart camera with FPGA based image analysis system dedicated to real-time tracking of a moving object," Journal of Systems Architecture, 2015.

[25] HARRIS, C and STEPHENS, "A combined corner and edge detector," presented at the Vision Conference, 1988, pp. 147–151.

[26] I. Bravo, M. Mazo, J. L. Lázaro, A. Gardel, P. Jiménez, and D. Pizarro, "An Intelligent Architecture Based on Field Programmable Gate Arrays Designed to Detect Moving Objects by Using Principal Component Analysis," Sensors, vol. 10, no. 10, pp. 9232–9251, Oct. 2010, doi: 10.3390/s101009232.

[27] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey California USA: ACM, Feb. 2015, pp. 161–170. doi: 10.1145/2684746.2689060.

[28] L. Bai, Y. Zhao, and X. Huang, "A CNN Accelerator on FPGA Using Depthwise Separable Convolution," IEEE Trans. Circuits Syst. II, vol. 65, no. 10, pp. 1415–1419, Oct. 2018, doi: 10.1109/TCSII.2018.2865896.

[29] Z. Wei, D.-J. Lee, and B. E. Nelson, "FPGA-based Real-time Optical Flow Algorithm Design and Implementation," JMM, vol. 2, no. 5, pp. 38–45, Sep. 2007, doi: 10.4304/jmm.2.5.38-45.

[30] Horn, B.K. and Schunck, "Determining optical flow," 1981, pp. 185–203.

[31] F. Barranco, M. Tomasi, J. Diaz, M. Vanegas, and E. Ros, "Parallel Architecture for Hierarchical Optical Flow Estimation Based on FPGA," IEEE Trans. VLSI Syst., vol. 20, no. 6, pp. 1058–1067, Jun. 2012, doi: 10.1109/TVLSI.2011.2145423.

[32] I. Ishii, T. Taniguchi, K. Yamamoto, and T. Takaki, "High-Frame-Rate Optical Flow System," IEEE Trans. Circuits Syst. Video Technol., vol. 22, no. 1, pp. 105–112, Jan. 2012, doi: 10.1109/TCSVT.2011.2158340.

[33] H.-S. Seong, C. E. Rhee, and H.-J. Lee, "A Novel Hardware Architecture of the Lucas–Kanade Optical Flow for Reduced Frame Memory Access," IEEE Trans. Circuits Syst. Video Technol., vol. 26, no. 6, pp. 1187–1199, Jun. 2016, doi: 10.1109/TCSVT.2015.2437077.

[34] C. Wang, E. D. Burnham-Fay, and J. D. Ellis, "Real-time FPGA-based Kalman filter for constant and non-constant velocity periodic error correction," Precision Engineering, vol. 48, pp. 133–143, Apr. 2017, doi: 10.1016/j.precisioneng.2016.11.013.

[35] P. Babu and E. Parthasarathy, "FPGA implementation of multi-dimensional Kalman filter for object tracking and motion detection," Engineering Science and Technology, an International Journal, vol. 33, p. 101084, Sep. 2022, doi: 10.1016/j.jestch.2021.101084.

[36] M. Genovese, E. Napoli, D. De Caro, N. Petra, and A. G. M. Strollo, "FPGA Implementation of Gaussian Mixture Model Algorithm for 47 fps Segmentation of 1080p Video," Journal of Electrical and Computer Engineering, vol. 2013, pp. 1–8, 2013, doi: 10.1155/2013/129589.

[37] G. K. Gultekin and A. Saranli, "An FPGA based high performance optical flow hardware design for computer vision applications," Microprocessors and Microsystems, vol. 37, no. 3, pp. 270–286, May 2013, doi: 10.1016/j.micpro.2013.01.001.

[38] S. Guo et al., "A system on chip-based real-time tracking system for amphibious spherical robots," International Journal of Advanced Robotic

Systems, vol. 14, no. 4, p. 172988141771655, Jul. 2017, doi: 10.1177/1729881417716559.

[39] J. G. Pandey, "An embedded FPGA-SoC framework and its usage in moving object tracking application," Des Autom Embed Syst, vol. 25, no. 3, pp. 213–236, Sep. 2021, doi: 10.1007/s10617-021-09252-y.

[40] M. Genovese, E. Napoli, D. De Caro, N. Petra, and A. G. M. Strollo, "FPGA Implementation of Gaussian Mixture Model Algorithm for 47 fps Segmentation of 1080p Video," Journal of Electrical and Computer Engineering, vol. 2013, pp. 1–8, 2013, doi: 10.1155/2013/129589.

الخوارزميات القائمة على الدوائر القابلة للبرمجة والمطمورة لتتبع الاشياء: مراجعة بحثية

مروان عبدالخالق ذنون * **سعد أحمد القزاز ****

marwanathy1972@umosul.edu.iq kazzazs60@umosul.edu.iq

* قسم الهندسة الكهربائية، كلية الهندسة، جامعة الموصل، الموصل، العراق
** قسم هندسة الميكاترونكس، كلية الهندسة، جامعة الموصل، الموصل، العراق

تاريخ الاستلام: 13 مارس 2023 استلم بصيغته المنقحة: 20 ابريل 2023 تاريخ القبول: 27 ابريل 2023

الملخص

أصبحت أنظمة الكشف عن الأشياء وتتبعها ذات أهمية متزايدة في التطبيقات العملية في الوقت الحاضر, أدى ذلك إلى اهتمام العديد من المجاميع البحثية من أجل تطوير وتحسين أداء مثل هذه الأنظمة ، حيث تم تطوير واقتراح العديد من الأساليب المتنوعة في هذا المجال. إن عملية تطوير وتنفيذ الرؤية الحاسوبية مستمرة باستخدام الأنظمة القابلة للبرمجة والأنظمة المدمجة. إن الهدف من هذه الدراسة هو تقديم ماتم اقتراحه وتطويره من الأبحاث السابقة والحديثة في مجال أنظمة الكشف والتتبع باستخدام رؤية الحاسوب التي تستخدم منصات FPGA *و* FPGA-SoC *تتضمن الدراسة وصفًا موجزًا للعديد من الخوارزميات الشائعة وخصائصها الأساسية والمجال الذي يفضل استخدامها فيه. كما تم تقديم مقارنة في استخدام الموارد وأيها الأكثر أو الأقل استخداما لتنفيذ الخوارزميات المختلفة في المجال المذكور. حيث وجدت الدراسة أن دوائر الـ* flip-flops (FF) *والـ (LUT) هي شائعة الاستخدام في تنفيذ أغلب الخوارزميات، في حين أن* BRAM *و* DSP *والمضاعفات (Multipliers) كانت لها أقل نسبة استخدام. نتيجة التطور الهائل في إنتاج أنظمة المعالجة الرقمية المتقدمة ،أصبح واضحا بأن هناك زيادة في التركيز على استخدام منصات* FPGA-SoC *في أنظمة المراقبة المرئية. والسبب وراء ذلك هو قدرتها على تنفيذ المعالجة المعقدة باستخدام التصميم المشترك البرمجي والمادي للحصول على أداء عالٍ في وقت أقل مقارنة مع استخدام الـ*FPGA .

الكلمات الداله :

طرح خلفية الصورة ; خوارزمية الكشف ; معدل نقل البيانات ; خوارزمية التتبع ; موارد البوابات القابلة للبرمجة حقليا