Hindawi Publishing Corporation
Modelling and Simulation in Engineering
Volume 2015, Article ID 948960, 8 pages
http://dx.doi.org/10.1155/2015/948960

Research Article

A Comparative Study of Multiple Object Detection Using Haar-Like Feature Selection and Local Binary Patterns in Several Platforms

Souhail Guennouni,[1] Ali Ahaitouf,[1] and Anass Mansouri[2]

[1]*Sidi Mohammed Ben Abdellah University, Faculty of Science and Technology, Renewable Energy and Smart Systems Laboratory, BP 2202, 30000 Fez, Morocco*
[2]*Sidi Mohammed Ben Abdellah University, National School of Applied Sciences, Renewable Energy and Smart Systems Laboratory, BP 72, 30000 Fez, Morocco*

Correspondence should be addressed to Souhail Guennouni; mrsouhail@gmail.com

Received 7 October 2015; Accepted 2 December 2015

Academic Editor: Aiguo Song

Object detection has been attracting much interest due to the wide spectrum of applications that use it. It has been driven by an increasing processing power available in software and hardware platforms. In this work we present a developed application for multiple objects detection based on OpenCV libraries. The complexity-related aspects that were considered in the object detection using cascade classifier are described. Furthermore, we discuss the profiling and porting of the application into an embedded platform and compare the results with those obtained on traditional platforms. The proposed application deals with real-time systems implementation and the results give a metric able to select where the cases of object detection applications may be more complex and where it may be simpler.

1. Introduction

Object detection is meant to detect the specific location and size of a particular object in an image or a video scene. With the growing need of detection-based security and industrial applications, the object detection in a fast and reliable manner has been attracting much interest.

There are two types of applications that can use this system. Applications which do not need real-time responses such as surveillance cameras to detect certain shapes like animal shapes in a hospital and applications which require real-time response like fire arms detection in an airport, for example, will require a better performance in terms of detection accuracy and response time. The strength of this system is that it can be trained for any type of object to be detected for different situations. An extension to this work would be to adapt the system to a low cost card and adapt it to the card architecture in order to get better performance and to be able to meet real-time requirement.

There are many types of object detection. One of them is knowledge based methods, which rely on a group of rules of object structures based on the relationship between the features of the specific object to be detected [1]. The second type is template matching based methods. In the case of face detection, the face templates are stored. The detection face is based on the liaison between the images and the stored template [2]. The third type of object detection is based on machine learning methods. This kind of methods requires a learning phase [3]. The algorithm needs to be trained to detect the specific object from a set of images. Then it can be used to detect the object.

Achieving high performance and a near-real-time object detection is a key concern in both large-scale systems and embedded platforms. Therefore, a reliable and accurate near-real-time object detection application, running on an embedded system, is crucial, due to the rising security concerns in different fields. Cascade classifier is one of the fewest algorithms to run in real-time. The key to its

high performance is the use of integral image, which only performs basic operations in a very low processing time. However, cascade classifier can only perform in a fixed aspect ratio. There were many attempts to respond to real-time constraints for object detection. Viola and Jones [3] have come up with a method of rectangular Haar-like features with AdaBoost learning algorithm combined with a cascade of strong classifiers. Then Leinhart and Maydt [4] improved the set of Haar-like features. Then, in [5], Ahonen et al. applied Local Binary Patterns for face detection and obtained good results compared to Haar-like results.

The proposed object detection application can be deployed in different platforms; it can be deployed on a high performance platform as well as in mobile platform. It can also be used in surveillance systems with distributed cameras and a back-end server in which the detection takes place. It can also be used in mobile devices equipped with camera and processor. A highly short response time in terms of detection is essential for such systems.

In a previous work [6] we have reported on the implementation of object detection using Haar-like feature selection using OpenCV for an embedded platform. And in this paper, we develop an extension to compare two feature selection algorithms which are the Viola and Jones detection technique based on Haar-like feature selection combined with cascade classifier, and the LBP (Local Binary Patterns) for feature selection combined with the same classifier, in terms of performance and accuracy in both standard platform and embedded platform.

2. Related Work

There are many publications related to object detection but the main challenge remaining in most of the published work is accuracy. However, there were many accomplishments in terms of the speed and precision of object detection. Areas of improvements were focusing mainly on feature selection and classification. Different types of feature selection algorithms were used. Mainly Haar features [3] were initially introduced for face detection and Local Binary Patterns (LBP) [5].

One research trend in object detection is to combine multiple sources of information like color and motion [7]. Having more information sources enhances the detection quality but in a significant detection time. In terms of classifiers, SVM (Support Vector Machine) classifier is popular due to its speed of detection.

A more advanced version of SVM was introduced by Felzenszwalb et al. [8] which is basically a framework for object detection which has been applied to a large number of objects types. Another sophisticated classifier was introduced first by Viola and Jones [3] to detect faces and was then generalized to detect other object types.

Gall et al. [9], using generalized Hough transform for object detection, achieved very interesting results in terms of accuracy in benchmark detection dataset.

Another research orientation proposes to add more resources in terms of hardware. Having parallel CPUs was proven to give good results in real-time detection. Parallelizing algorithm processing in different CPUs is an efficient way to enhance the algorithm performance on a specific platform [10, 11].

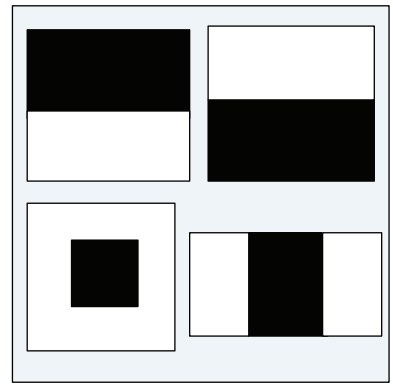

Figure 1: Rectangular feature.

The cascade classifier [3], adopted here for object detection, is based on Haar-like feature and consists of combining many complex classifiers in cascade structure, which increases the speed of object detection.

2.1. Haar-Like Feature. Haar-like feature's principle is based on detection of features encoding of some information about the class to be detected. They are adjacent rectangles in a particular position of an image. Figure 1 shows the types of Haar-like features depending on the number of adjacent rectangles. According to [4] there are three types of Haar-like features. The first type is the edge feature, which is represented in Figure 1 by the two upper squares. The second type is the line feature (Figure 1 lower left square). The last type of features is the center-surround feature (Figure 1 lower right square).

As described in [6], the idea behind Haar-like feature selection algorithm is simple. It lies on the principle of computing the difference between the sum of white pixels and the sum of black pixels. The main advantage of this method is the fast sum computation using the integral image. It is called Haar-like because it is based on the same principle of Haar wavelets [4].

2.2. Integral Image. Rectangle features can be computed rapidly using an intermediate representation of the image called the integral image [3]. The integral image consists of having small units' representation of a given image.

For example (see Figures 2 and 3), the value of this integral image at location 1 is the sum of pixels in rectangular A. The value at location 2 is $A + B$ and so on. So the sum of pixels in rectangular D is

$$\text{sum}(D) = ii(4) - ii(3) - ii(2) + ii(1), \quad (1)$$

where sum(D) is the sum of pixels in the rectangular D only, which is the sum of pixels in the rectangle $A + B + C + D$, represented by $ii(4)$, minus $ii(3)$, which is the integral image of rectangle $A + C$, minus $ii(2)$, which is the integral image of $A + B$, and finally we had $ii(1)$, which is the integral image of

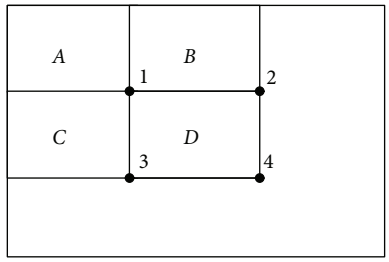

Figure 2: Integral image blocks.

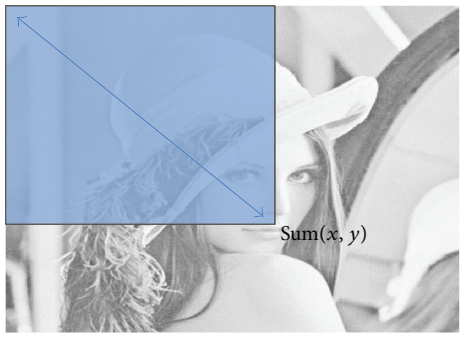

Figure 3: Integral image sum.

the rectangle A (the addition is performed because the region A is subtracted twice in $ii(3)$ and $ii(2)$).

The example in Figure 4 shows the integral transformation of images. The blue matric represents the original image, and the purple one represents the integral image. The integral value of the shaded part in the first matrix is 20, which is the sum of pixel values in the shaded area, starting from the upper left pixel of the image. In the same manner, the integral image of the shaded area in the third matrix is 17. It can be easily calculated in the fourth matrix by subtracting the integral image up to pixel 3, minus the integral image of the regions outside the shaded area starting from the upper left pixel. Therefore, the integral image of the shaded area is 34, which represents the shaded area up to pixel 3, minus 14, which is the integral image of the region above the shaded area, minus 8, which is the integral image of the region on the left of the shaded area, plus 5, which was included in both integral value subtractions (14 and 8), which yields $34 - 14 - 8 + 5 = 17$.

The integral image is defined as follows:

$$ii(x, y) = \sum_{x' \le x, y' \le y} i(x', y'), \tag{2}$$

where $ii(x, y)$ is the integral image and $i(x', y')$ is the original image.

Therefore, the integral value of a specific pixel is the sum of pixels on the top of it towards the left [3, 12]. Then the image can be integrated in fewer pixel operations, since the traversing starts on the top left towards the bottom right.

2.3. The Local Binary Patterns

2.3.1. Local Binary Patterns Operator. The Local Binary Pattern (LBP) is a simple and efficient texture operator, which labels the pixels of an image by thresholding the neighboring of each pixel, resulting in a binary number as shown in Figure 6. It was initially introduced by Ojala et al. [13]. For each pixel, the algorithm considers the 8 neighboring pixels. Then based on the gray-scale value of the selected pixel, it assigns the neighboring pixels the value of 0 or 1. Therefore, every pixel will have a string of binary values. Figures 4 and 5 show an example of the calculation.

In 2002, Ojala et al. [13] extended their original LBP operator to a circular neighborhood of different radius size. Another extension to the original operator is the definition of uniform patterns, which can be used to reduce the length of the feature vector and implement a simple rotation-invariant descriptor.

A Local Binary Pattern is called uniform if the binary pattern contains at most two bitwise transitions from 0 to 1 or vice versa when the bit pattern is traversed circularly. For example, the patterns 00000000 (0 transitions), 01110000 (2 transitions), and 11001111 (2 transitions) are uniform whereas the patterns 11001001 (4 transitions) and 01010010 (6 transitions) are not.

2.3.2. LBP Detection Algorithm. The algorithm used is a variant of the cascade algorithm introduced by Viola and Jones [3] and uses LBP features instead of Haar-like features in order to have faster processing and boosted classifiers. The LBP algorithm slides its processing window over the object image for evaluating the successive stages of the cascade algorithm by scoring their features. Each feature is described by 3×3 neighboring rectangular areas.

The value of each feature is computed by comparing the central area with the neighboring area around it (8 neighbors). The result is in the form of a 8-bit binary value called LBP. A number of features represent a stage of the cascade algorithm. Every feature has positive and negative weights associated with it. For the case where the feature is in consistence with the object to be detected, the positive weight is added to the sum. For the case where the feature is inconsistent with the object, the negative value is added to the sum. The sum is then compared to the threshold of the stage. If the sum is below the threshold, the stage fails and the cascade terminates early, and, thus, the processing window moves to the next window. If the sum is above the threshold, the next stage of the cascade is attempted. In general, if no stage rejects a candidate window, it is assumed that the object has been detected.

In order to avoid the redundancy of computing the integral of rectangles, the integral images are calculated to speed up the calculation of the feature.

2.4. AdaBoost Learning Algorithm. Viola and Jones [3] used AdaBoost learning algorithm to select a specific Haar-like feature as a threshold. AdaBoost is used to create strong classifier from combining a collection of weak classification functions [11].

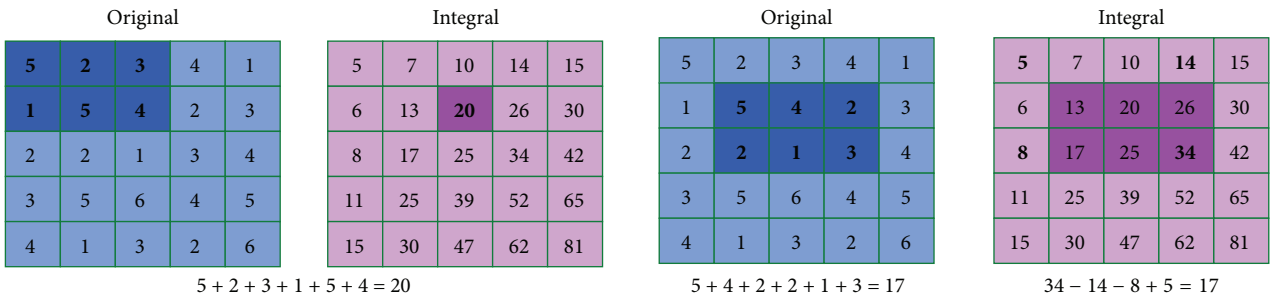

Figure 4: Integral image: illustration example.

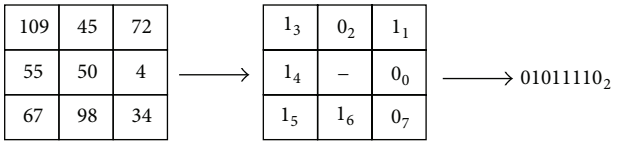

Figure 5: LBP calculation example.

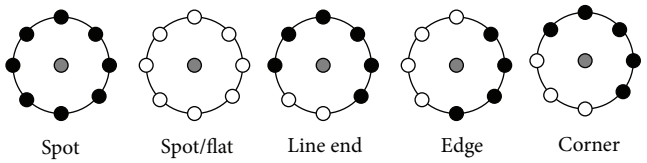

Figure 6: Example of texture primitives detected by LBP.

The strongest classifier uses the strongest feature, which is the best Haar-like feature, that is, the feature that best separates the positive and negative samples.

2.5. Cascade Classifier. Cascade classifier [3] is a chain of weak classifiers for efficient classification of image regions. Its goal is to increase the performance of object detection and to reduce the computational time. As shown in Figure 7, each node in the chain is a weak classifier and filter for one Haar feature. AdaBoost gives weights to the nodes, and the highest weighted node comes first.

When a filter fails to pass image regions, that specific subwindow of the image is eliminated for further processing. It is then considered as a nonobject. Meaning that the image regions processed do not contain the object to be detected. This is very crucial to the performance of the classifier, since all or nearly all negative image subwindows will be eliminated in the first stage. On the other hand, when image regions successfully passed the filter, they go to the following stage, which contains a more complex filter. Only regions that successfully pass all filters are considered to contain a match of the object. This means that regions of the image contain the object subject to detection.

The reason behind the multistage classifier is to reject efficiently and rapidly the nonobject subwindows. The next nodes in the chain in Figure 7 represent complex classifiers, in the case of face detection. The classifier is used to reject more false positives (nonface regions) of the subwindows [3].

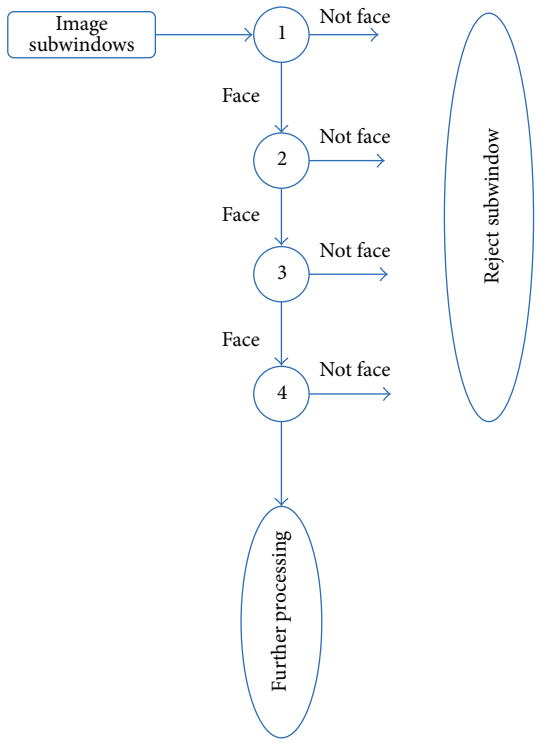

Figure 7: Cascade classifier.

The number of false positives is radically reduced after several steps of processing.

3. Implementation and Results

3.1. Standard Platform. The object detection system has been developed on PC Xeon-based servers (E5670 clocked at 2.93 GHz) and using gcc 4.4.5.

The training phase was performed on this machine for different objects. Thus, different xml files were generated for each object.

The detection was performed on a desktop platform as a reference result carried out using a standard midrange camera and using both live pictures taken and images given to program. Those images contain both objects subject to training and other random objects. The result of detection is displayed on the screen of the computer.

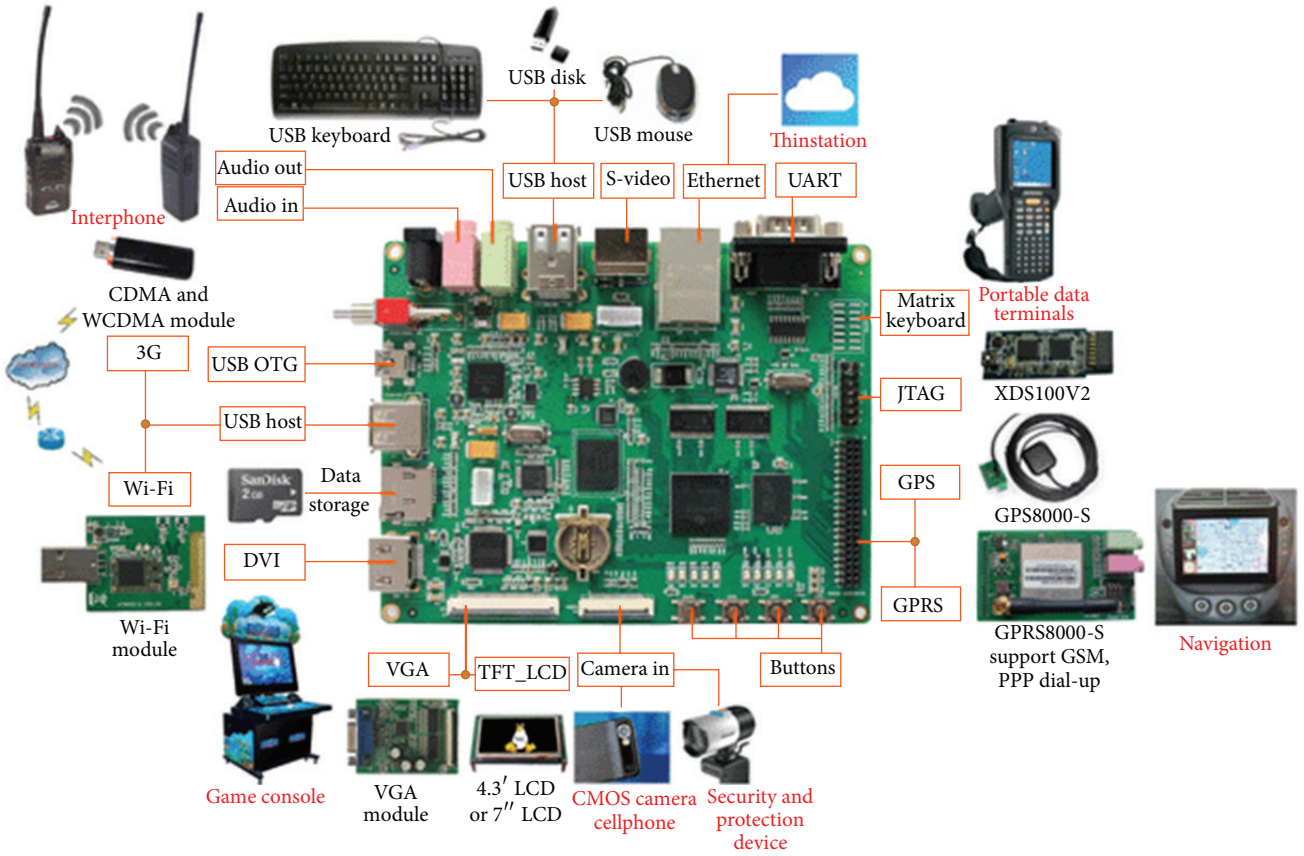

Figure 8: DM3730 board.

3.2. Embedded Platform. The embedded system used in this work is the Texas Instrument DM3730 digital media processor, Figure 4. It has 1 GHz ARM Cortex-A8 processor and a DSP core. It supports hardware video accelerator enabling HD 720p video decoding and encoding independent of the ARM processor. It contains a USB 2.0 slot used as an input for detection. It has 512 Mbytes of RAM. The system is used through a mouse and keyboard on the card connected through USB ports (Figure 8).

The operating system (OS) is the embedded Linux version Angstrom. The system files, for the processor booting, are stored in the flash-NAND memory.

3.3. Training Dataset. For the training of the cascade classifier, concerning the detection of a particular object, we use a set of images coming from the object itself.

To achieve a high detection rate of the object, we needed to use a large number of images in the training phase. The number of images we used as training set of a particular object is around 4000 positive images. The positive images are images that include the targeted object among others. Other images are also used (negative images) that do not include the object for the training phase.

The dataset we used includes images of the object from different angles in order to make the detection possible in most angles.

The images of objects we used in this application include face object, hand object, and pedestrian (human body object).

For face detection, we used FEI face database [14], which consists of a large number of face images from different angles and in different positions. For other objects, we created a database of images extracted from video capture of objects subject to training from all angles and positions.

The output of the cascade training is an xml file that contains data about the object to be detected. An xml file is generated for each object to be detected. The xml file is then used by our application in order to perform the detection. The training was performed separately for both algorithms.

3.4. Implementation of the Proposed Application. The application implementation was performed using C++ language using OpenCV libraries. The compilation was performed using GCC (GNU Compiler Collection).

You need GNU project C and C++ compiler for compiling C program and creating an executable file for the target platform.

The platform included in the embedded Linux environment includes a prebuilt SD card image from which you can boot and run cross-compiled application code. When you make code changes, it is valuable to rerun a software-only compilation to verify that your changes did not adversely change your program.

Table 1: Detection function parameters.

scaleFactor	2.0
minNeighbors	4

Table 2: Consumed time for each platform.

Platform	Haar-like feature based cascade algorithm (ms) [6]	LBP feature based cascade algorithm (ms)
Standard platform	31	38
Texas instrument's DM3730	95	90

The cascade classifier detection function "detectMultiScale" was given the parameters in Table 1.

The detection function uses the two listed parameters in Table 1, where the scale factor determines the possible object size which is related to how far the object is from the camera, and the minimum neighbors are the minimal number of hits needed to detect the object. That is, if there is less detection than "minNeighbors" the object detected will be discarded.

3.5. Results. The comparison of the two algorithms is performed in two steps.

The first one represents the measurement of the performance of each algorithm for the detection of a single object in a specific scene.

The second one compares the performances of the two algorithms in detecting multiple objects in a specific scene.

Figure 9 shows the results of object detection for different objects. The first image is the result of face detection, the second one is the result of pedestrian object detection, and third image shows the results of hand gesture detection.

Comparison of Haar and LBP Performance Results for a Single Object

Results Analysis. Table 2 shows the results comparison of the performances of Haar-like and LBP algorithms in both platforms for a single object detection.

As generally expected, each algorithm performance on the regular platform is better than in an embedded platform. As shown in Table 2, the execution time in the standard platform is smaller than the execution time in the embedded system. The Haar-like based cascade is better on the standard platform than LBP based cascade.

However, in the embedded platform we can clearly notice that LBP is performing better than Haar-like feature, in terms of detection time. This shows that the LBP algorithm performs better under limited resources, and Haar-like feature algorithm performs better on the regular platform where there is more resource availability.

In terms of accuracy, Haar-like performance is shown to be better than LBP in the standard platform. It has accuracy rate of 96.24% versus 94.75% for LBP. On the embedded platform, we also notice that Haar-like accuracy is slightly better than LBP with a hit rate of 93.56% and 92.65% for LBP.

Table 3: Consumed time for each platform for Haar-like algorithm.

Platform		Execution time (ms): 1 object	Execution time (ms): 2 objects	Execution time (ms): 3 objects
Standard platform	HL	31	35	43
	LBP	38	45	52
Texas instrument's DM3730	HL	95	99	110
	LBP	89	94	97

This percentage is computed based on the ratio of hit, miss, and false detection rates of the objects to be detected.

Comparison of Performance Results for Simultaneous Detection of Multiple Objects Using Both Algorithms. Table 3 shows the performance of each algorithm for detecting simultaneously multiple objects in a given scene.

The overall performances in the standard platform for multiple objects are better than those on the board. However, we notice that the detection system on the board is stable when we increase the detection.

The results given by the embedded system can be considered as positive and encouraging. In fact, we can notice that, for multiple detection, the increase of the consumed time compared to the standard platform is limited for each algorithm. Actually, the increased consumed time seems to be even smaller for the LBP algorithm in the embedded platform. This makes the feasibility of using the system on an embedded platform more realistic.

Figure 10 shows the comparison between the two platforms for each algorithm.

Results Discussion. The results of LBP and Haar-like algorithms in multiple detection are competitive and encouraging in general. The real-time constraints are only reached in the standard platform for the moment.

However, for the embedded platform, some enhancements on the algorithms still needed to achieve real-time constraints.

On the other hand, as it was the case for the first set of results, LBP gives a better performance in the embedded system. This proves that it performs better in a limited resources environment. Haar-like algorithm gives better results than LBP in the standard platform, for both single object detection in a scene and multiple object detection. This proves that Haar-like algorithm needs more resources than what is offered by the system to be able to perform better.

For all cases, the profiling on the embedded platform allowed us to detect blocs of the algorithms that consume the most processor time. The result helps to identify the algorithms blocs that need enhancements in order to achieve better results on the embedded platform. We can actually run independent execution tasks in parallel processors in order to enhance the performance of detection to achieve real-time constraints.

Figure 9: The results of object detection using both algorithms.

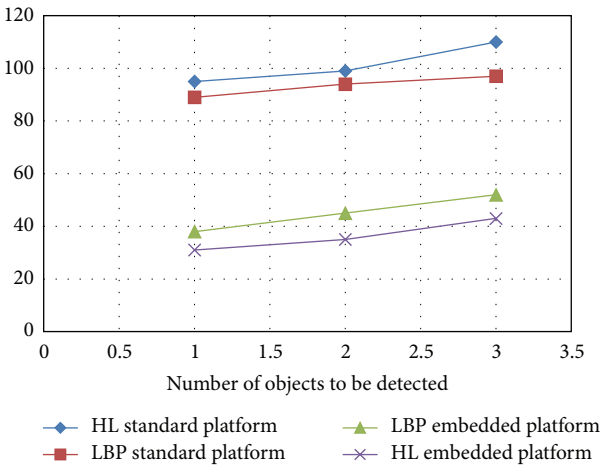

Figure 10: Performance on different platforms.

4. Conclusion and Future Work

In this paper we discuss the performance of two feature selection algorithms Haar-like feature selection and LBP for the detection of a single object and multiple objects in the same scene and for both standard platform and embedded system.

The results illustrated above help us to determine which algorithm can be more efficient in the different environment.

We have combined Haar-like feature selection with cascade classifier and LBP with cascade classifier for accurate comparison.

The goal is to enhance the performance on a low resource embedded system to meet real-time constraints.

From the results above we can see that on the standard platform both algorithms performances meet or are close to real-time object detection.

The next steps of this work will be to enhance the embedded platform performance. This enhancement can be achieved through the usage of parallelism. We can get several processors to run simultaneously separate tasks in order to enhance performance and response time.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] S. Z. Li and A. K. Jain, *Handbook of Face Recognition*, Springer, 2005.

[2] A. L. Yuille, P. W. Hallinan, and D. S. Cohen, "Feature extraction from faces using deformable templates," *International Journal of Computer Vision*, vol. 8, no. 2, pp. 99–111, 1992.

[3] P. Viola and M. J. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[4] R. Leinhart and J. Maydt, "An extended set of haar-like features for rapid object detection," in *Proceedings of the International Conference on Image Processing (ICIP '02)*, vol. 1, pp. I-900–I-903, IEEE, Rochester, NY, USA, September 2002.

[5] T. Ahonen, A. Hadid, and M. Pietikäinen, "Face description with local binary patterns: application to face recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 2037–2041, 2006.

[6] S. Guennouni, A. Ahaitouf, and A. Mansouri, "Multiple object detection using OpenCV on an embedded platform," in *Proceedings of the 3rd IEEE International Colloquium in Information Science and Technology (CIST '14)*, pp. 374–377, IEEE, Tetouan, Morocco, October 2014.

[7] B. Heisele, U. Kressel, and W. Ritter, "Tracking non-rigid, moving objects based on color cluster flow," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '97)*, pp. 257–260, San Juan, Puerto Rico, June 1997.

[8] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[9] J. Gall, A. Yao, N. Razavi, L. Van Gool, and V. Lempitsky, "Hough forests for object detection, tracking, and action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 11, pp. 2188–2202, 2011.

[10] W. Bing and C. Chareonsak, "FPGA implementation of AdaBoost algorithm for detection of face biometrics," in *Proceedings of the IEEE International Workshop on Biomedical Circuits and Systems*, pp. 17–20, Singapore, December 2004.

[11] M.-T. Pham and T.-J. Cham, "Fast training and selection of Haar features using statistics in boosting-based face detection," in *Proceedings of the 11th International Conference on Computer Vision (ICCV '07)*, pp. 1–7, IEEE, Rio de Janeiro, Brazil, October 2007.

[12] R. Benenson, M. Mathias, R. Timofte, and L. Van Gool, "Pedestrian detection at 100 frames per second," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '12)*, pp. 2903–2910, Providence, RI, USA, June 2012.

[13] T. Ojala, M. Pietikäinen, and T. Mäenpää, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, 2002.

[14] "FEI Face Database," http://fei.edu.br/~cet/facedatabase.html.