Multiple Object Detection using OpenCV on an Embedded Platform

Souhail Guennouni
Sidi Mohammed Ben Abdellah University
School of Science and Technology
Signals Systems and Components Laboratory
B.P. 2202, 30000 Fez, Morocco
mrsouhail@gmail.com

Anass Mansouri
Sidi Mohammed Ben Abdellah University,
National School of Applied Sciences,
Signals Systems and Components Laboratory
BP 72 Fez, Morocco

Ali Ahaitouf
Sidi Mohammed Ben Abdellah University
School of Science and Technology
Signals Systems and Components Laboratory
B.P. 2202, 30000 Fez, Morocco

Abstract—**Object detection has been attracting much interest due to the wide spectrum of applications that use it. Object detection technology has been driven by an increasing processing power available in software and hardware. 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 comparing the results with the regular platform. The proposed application deals with real time systems implementation and the results give an indication of where the cases of object detection applications may be more complex and where it may be simpler.**

Keywords-component; OpenCV; image processing; object detection; embedded system

I. INTRODUCTION

With the advancement in the video surveillance and image processing, object detection has known a rising interest in the computer visualization industry. However, 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. The application can be deployed in different platforms; it can be deployed on a high performance platform as well as in mobile platform. This application can be used in surveillance systems with distributed cameras and a backend server in which the detection takes place. It can also be used in mobile devices equipped with camera and processor. A high response time in terms of detection is essential for such systems.

We are particularly interested in designing a system able to simultaneously detect multiple objects on a scene. This detection information will help surveillance cameras send real-time information about the detected objects to the back end central system. The information sent to the back end system can be used to detect the presence of an object in the covered area or to recognize a specific person (blacklisted person) from the detected face in the area.

In particular, we are able to train our application in order to detect any object the surveillance system is interested in such as guns or dogs…

This paper presents an overview of the cascade object detection algorithm as well as Haar-Like feature selection used by cascade classifier. Then, it proposes an OpenCV-based solution for multiple object detection, and finally, presents the results of the comparison of performances in a regular platform and an embedded device.

The detection principle used in the application is based on object detection proposed by Viola et al. [4] for facial detection where the Haar-like feature based cascade classifier for the detection, is adopted.

The OpenCV library is described in section II then the adopted object detection is detailed in section III. Section IV presents the system, the developed application and the results obtained after porting the application on regular platform and embedded platform.

II. OPEN CV

OpenCV [1] is an open source computer vision library that is used in real time computer vision. OpenCV was developed by Intel and now supported by Willow Garage and Itseez. OpenCV is designed and optimized for real time applications, although it's developed in C and C++ languages, it's a cross-platform library that runs on Linux, Windows and Mac OS [1]. The OpenCV library contains hundreds functions that cover many areas in computer vision such as robotics, medical image processing, security [2]

III. OBJECT DETECTION

The cascade classifier [4], adopted here for object detection is based on Haar-like feature and is consists of combining many complex classifiers in cascade structure, which increases the speed of object detection.

978-1-4799-5979-2/14/$31.00 ©2014 IEEE

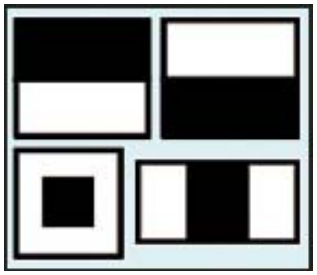

Fig 1: Rectangular feature

A. 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 [9] 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, which is represented in figure 1 by the lower right square. The last type of features is the center-surround feature, which is represented in figure 1 lower left square.
The Haar-like principle is simple; it lies on computing the difference between the sum of white pixels and sum of black pixels. The main advantage of this method is the fast sum computation using the integral image. It's called Haar-like because it's based on the same principle of Haar wavelets [3].

B. Integral Image

Rectangle features can be computer rapidly using an intermediate representation of the image called the integral image [4]. The integral image consists of having small units' representation of a given image.

For example (see figure 2), 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:

$$sum(D) = ii(4) - ii(3) - ii(2) + ii(1) \qquad (1)$$

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 the ii(1), which is the integral image of the rectagle A (the addition is performed become the region A is subtracted twice in ii(3)and ii(2)).

The integral image is defined as follows:

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

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 [4], [5]. Then the image can be integrated in fewer pixel operations, since the traversing starts on the top left towards the bottom right.

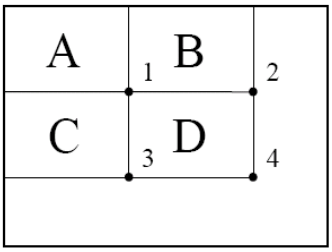

Fig 2: Integral Image

C. AdaBoost Learning Algorithm:

Viola and Jones [4] 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 [6].
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.

D. Cascade Classifier:

Cascade classifier [4] 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 3, 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 [7].
When a filter fails to pass image regions, that specific sub-window of the image is eliminated for further processing. It is then considered as a non-object. 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 sub-windows 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 multi-stage classifier is to reject efficiently and rapidly the non-object sub-windows. The next nodes in the chain in Fig 3 represent complex classifiers, in the case of face detection. The classifier is used to reject more false positives (non-face regions) of the sub-windows [8]. The number of false positives is radically reduced after several steps of processing.

978-1-4799-5979-2/14/$31.00 ©2014 IEEE

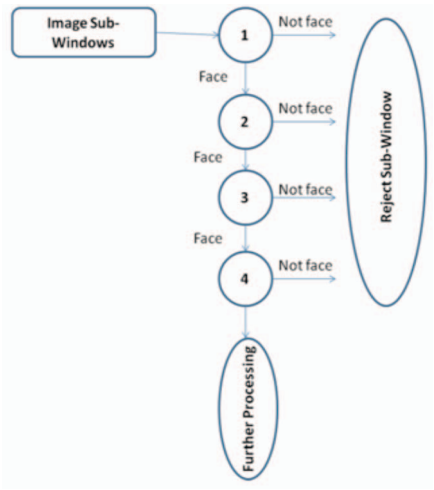

Fig 3: Cascade Classifier

IV. IMPLEMENTATION AND RESULTS

A. *Standard Platform*

The object detection system has been developed on a windows machine, with the following characteristics:

- Intel Processor Core 2 Duo
- 4 GB of RAM

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

The detection was performed on a desktop platform as a reference result. The detection was performed using a standard mid-range 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.

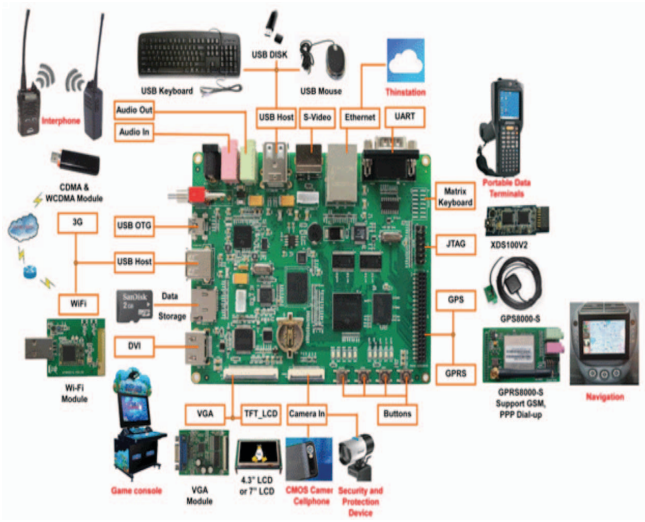

Fig 4: DM3730 board

B. *Emebedded platform:*

The embedded system used in this work is the Texas Instrument DM3730 digital media processor. It has 1GHz ARM Cotex-A8 processor and a DSP core. It supports hardware video accelerator enable 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.

A. *Training dataset*

The dataset used in training the cascade classifier in order to detect a particular object is a set of images of the object we want it to be detected later.

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 object in them. Other images are used (negative images) that do not include the object for in the training phase.

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

The images of objects we used in this application include objects like dog, hand signs, plastic bottle etc.

For face detection, we used FEI face database, 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 captures 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.

B. *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). The development and testing were done under Windows XP SP3. The first step was to test the object detection system under a windows machine, then port it on the card or other platforms.

The cascade classifier detection function "detectMultiScale" was given the following parameters:

TABLE 1: DETECTION FUNCTION PARAMETERS

scaleFactor	2.0
minNeighbors	4

Where, the scale factor determines the possible object size which is related to how far the object is from the camera, and minimum neighbors is minimal number of hits needed to detect the object. That is, if there are less detection than "minNeighbors" the object detected will be discarded.

C. Results

The execution time of cascade classifier for single object detection in both platforms is shown in the table below:

TABLE 2: CONSUMED TIME FOR EACH PLATFORM

Platform	**Time (ms)**
Windows based PC	*31 ms*
Texas Instrument's DM3730	*95ms*

The performance on the regular platform is better than in the embedded platform. This is shown in the table above by the execution time in the windows platform, which is the smaller than the execution time in the card.

Although these results were expected due to the difference in resources, the results in the embedded platform are encouraging.

The profiling on the embedded platform allowed us to detect blocs that consume more the processor time. The result helps to understand the blocs in the chain on the card that need enhancements in order to achieve better results on the embedded platform. Especially, when adding more processors to the card to perform specific tasks in order to enhance the performance to achieve acceptable results for real-time application.

V. CONCLUSION AND FUTURE WORK

The system developed in this work proves that object detection can be deployed in different platforms as needed. This system is a good match of the need to have surveillance cameras with object detection notification. An example is detecting firearms or animals in certain organizations or institutions.

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 performances.

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 be run simultaneously separate tasks in order to enhance performance and response time.

REFERENCES

[1] G. Bradski and, A. Kaehler, "Learning OpenCV", OReilly Publications, 2008

[2] M. Cerny, M. Dobrovolny, "Eye Tracking System on Embedded Platform", International conference of Applied Electronics 2012

[3] P. Aby, A. Jose, et al, "Implementation and optimization of an Embedded Face Detection system", International Conference on Signal Processing 2011.

[4] P. Viola, M. Jones, "Robust Real-Time Face Detection", International Journal of Computer Vision 57(2), 137154, 2004

[5] R. Gonzalez and R. Woods, "Digital Image Processing", PearsonEducation, 3rd Edition, 2008.

[6] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting" Journal of computer and system sciences, no. 55, pp. 110 – 140, 1997.

[7] W. Bing, and C. Chareonsak, "Fpga implementation of adaboost algorithm for detection of face biometrics", in Biomedical Circuits and Systems, 2004 IEEE International Workshop, Dec. 2004, pp. S1/6–17–20.

[8] M. Pham, T. Cham, "Fast training and selection of Haar features using statistics in boosting-based face detection", in: Proceedings of the IEEE International Conference on Computer Vision, 2007.

[9] R. Leinhart and J. Maydt, "An extended set of haar-like features for rapid object detection", in Proc. ICIP, 2002, vol.1, pp. 900 -910.