IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735.Volume 9, Issue 2, Ver. II (Mar - Apr. 2014), PP 73-79
www.iosrjournals.org

Implementation of Embedded ARM9 Platform using Qt And openCV For Human Upper Body Detection

Krunal Shantilal Patel[1], Prof. Kalpesh R Jadav[2]
[1,2]*Electronics and Comm. Dept. / Parul Institute of Eng. & Tech., Vadodara, India*

Abstract: In this Paper, A novel architecture for automotive vision using an embedded device will be implemented on ARM9 Board with highly computing capabilities and low processing power. Currently, achieving real-time image processing routines such as convolution, thresholding, edge detection and some of the complex media applications is a challenging task in embedded Device, because of limited memory. An open software framework, Linux OS is used in embedded devices to provide a good starting point for developing the multitasking kernel, integrated with communication protocols, data management and graphical user interface for reducing the total development time. To resolve the problems faced by the image processing applications in embedded Device a new application environment was developed. This environment provides the resources available in the operating system which runs on the hardware with complex image processing libraries. This paper presents the capture of an image from the USB camera, applied to image processing algorithms to Detect Human Upper Body. The application (GUI) Graphical User Interface was designed using Qt and ARM Linux gcc Integrated Development Environment (IDE) for implementing image processing algorithm using Open Source Computer Vision Library (OpenCV). This developed software integrated in mobiles by the cross compilation of Qt and the OpenCV software for Linux Operating system. The result utilized by Viola and Jones Algorithm with Haar Features of the image using OpenCV.

Keywords : *OpenCV, ARM Linux Integrated Development Environment, Qt Creator, Image Processing Library, Region of Interest, Embedded Platform, Viola-Jones, Haar Feature*

I. INTRODUCTION

The main difficulty facing in the advance of image processing applications such as in Embedded Device, which concerns the limited memory and CPU resources in order to store the images and the intermediate calculations, since there is limitations on the hardware resources available the operating system which runs on the hardware do not provide complex image processing libraries. Furthermore, today's Embedded Device doesn't contain committed arithmetic units for accomplishment of floating point operations which are inappropriately frequent in several applications [1]. ARM Board's animation and applications which include smaller display resolution, low quality image capture etc. is with embedded devices with additional challenges [2]. Most of the present ARM9 processor with Linux operating system with internal memory 64MB to 128MB [3]. This needs at least Linux Ubuntu 12.04LTS version, which offers provision for the Qt libraries. OpenCV is an open source library, which is widely used in image processing applications to attain instantaneous acknowledgement for embedded applications [4]. OpenCV library is written in C, confirming fast and convenient code, and which compiled for many embedded platforms. OpenCV library involves the elementary image tasks, which are logical & arithmetic operations. This also consists of complex operations, object detection and object tracking. Qt is a cross-platform application and UI framework that allows developers to write applications once and deploy them across many desktop and embedded operating systems without rewriting the source code. This paper describes how efficiently image processing applications can be run on Embedded Device. The results obtained demonstrate without loss of image information. The remainder of this paper is organized as follows: In Section 2, it provides previous works on Embedded Device. In Section.3, the simulation environment and the experimental methodology to evaluate the dynamic library functions are presented. In Section.4, the implementation of Embedded Device application using Qt is discussed in detailed. In Section 5, the extensive simulation results, analysis and the performance of the image quality are presented. Section 6 consists of the conclusion.

II. PREVIOUS WORKS ON EMBEDDED DEVICE

Since OpenCV library has many useful functions, which is widely used in many applications. Willow Garage's personal robot was developed using the OpenCV library with image Recognition n system [5]. Stanley also used OpenCV library for his image processing engine. J.-P. Farrugia et al. uses GPUCV library, it needs additional computational resources, quick processing and with extra power consumption [6]. Even though GPUCV library is group of OpenCV vision tasks, these can't use in embedded platforms such as mobiles, automotive and robots, which limits to thermal constraints. In Masayuki Hiromoto et al. paper gives embedded

applications includes mobile, robotics and etc. This also requires special embedded processors to achieve image recognition within the system's size and limited power [7]. Clemens Arth et al. was developed the specialized DSP based enhanced system for image recognition [8]. To overlay these issues on embedded applications, we propose optimized OpenCV implementation for image applications for mobile platforms.

III. METHODOLOGY AND SIMULATION ENVIRONMENT

In this section the tools and methodology to implement and evaluate the dynamically loading IPP with OpenCV libraries are detailed. The Intel Integrated Performance Primitives (IPP) was used to collect automatically OpenCV functions, to optimize the processors on the assembler level. So the task is to make those image processing libraries available on the development environment. This was performed with the image processing applications in real time. The computations on the processing are modeled as cycle accurate. The experimental results focused on the more complex comic images. We have used QT4.6.3 for the implementation of image application. This was developed by using OpenCV 2.0.0.CMAKE 2.8 GUI for cross compile OpenCV 2.0.0 for Ubuntu 12.04 LTS Express edition to get the image processing libraries. The simulation has done with Ubuntu 12.04 LTS on HP Laptop. HP Laptop built with Intel Duo Core CPU T5750 operated Linux OS, running at 2.0GHz with 32KB L1 D-Cache, 32KB L1 I-Cache 2MB L2 cache with 8-way set associative and 3GB RAM. Then using Qt, cross compiled images loads into ARM9 mini 2440 operated with Linux OS to display images without loss of quality of image.

IV. IMPLEMENTATION OF EMBEDDED PLATFORM USING OpenCV

The image processing algorithms are implemented using OpenCV. OpenCV was designed for high computational efficiency with a strong focus on real time applications. OpenCV was written with optimized C, so it could yield benefit of multi-core processors.

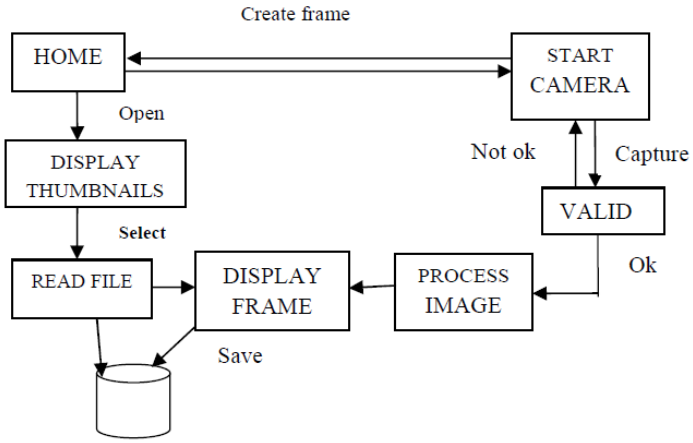

Figure1: Implementation of image processing on Hardware

The library was improved with C language according to mobile platform for image processing use on Windows. Fig. 1 shows the proposed implementation of image processing application on mobile. Opening of the application it will display home screen which has four options:

A. Create a strip

The application will start the camera and initiates it according to the .config file and has the standard buttons which are default to the handset, which basically includes capture button, settings and cancel button. The user can take picture by pressing capture. When the user takes picture the application will start all the image processing steps. If it succeeds the images will be stored in the specified directory in .config file and displays the images. If user press cancel button it opens the main screen. If the user press settings the camera settings will be opened and changed values will be written to .config file.

B. Open a strip

The application goes to the default directory and checks for the images and displays as list/thumbnails so the user can select the image. When image is selected it will display in full screen. It also has a button for going back to the image selection screen.

C. Help

Opens a file in the text edit box which is only readable where instructions for using the application is given

D. Exit:

Exit the application.

Proposed Flow of whole System

The figure 2 shows the proposed flow of the image processing application on a system. In this, first it startswith

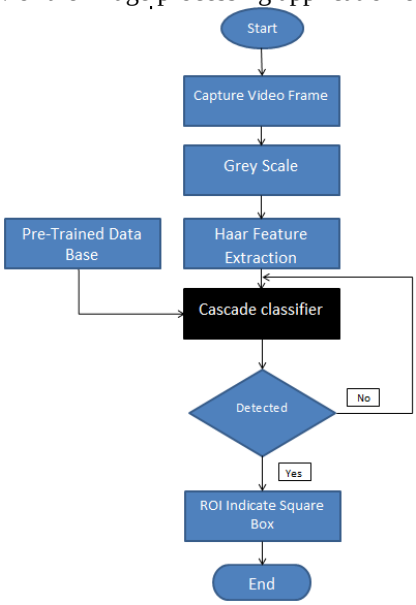

Figure 2: The flow of the Image processing application on Hardware

image capture, that captured image would be stored as temporary image in temporary buffers. Temporary image converted to grey scale using RGB to grey scale converter. Grey scale image now processed with Haar features by using Feature Extraction. Now detailed steps explanation given with individual blocks:

A. Capturing Image

The figure 3, demonstrates the algorithm of opening a new camera capture device to capture once frame, then open new window to display the result with the use of OpenCV library.

```
1. from opencv import highgui as hg
2. capture = hg.cvCreateCameraCapture
3. hg.cvNamedWindow(Snapshot)
4. frame = hg.cvQueryFrame(capture)
5. hg.cvShowImage(Snapshot, frame)
```

Figure 3: Image Capture and display with OpenCV

B. Grey Scale

An image, with each pixel value has single sample then it is known as Grey scale digital image, also it conveys only intensity information. Gray scale images are also called black-and-white, representing the nonexistence of any chromatic discrepancy. The figure 4, shows the algorithm of grey image conversion with OpenCV

```
# read image with gray scale effect
IplImage*grey_image=
cvLoadImage(IMAGE_PATH,CV_LOAD_IMAGE_GRAYSCALE);
#Show grey scale image
cvNamedWindow( "grey scale",
CV_WINDOW_AUTOSIZE) ;
cvShowImage( "grey scale",grey_image );
#Release image windows
cvReleaseImage( &grey_image );
```

Figure 4: Algorithm gray with OpenCV

C. Haar Feature

Haar features are composed of either two or three rectangles. Face candidates are scanned and searched for Haar features of the current stage. The weight and size of each feature and the features themselves are generated using a machine learning algorithm from AdaBoost [6][7]. The weights are constants generated by the learning algorithm. There are a variety of forms of features as seen below in Fig.5

(A) (B)

Figure 5: (A) Examples of Haar features. (B) Areas of white and black regions are multiplied by their respective weights and then summed in order to get the Haar feature value

Each Haar feature has a value that is calculated by taking the area of each rectangle, multiplying each by their respective weights, and then summing the results. The area of each rectangle is easily found using the integral image. The coordinate of the any corner of a rectangle can be used to get the sum of all the pixels above and to the left of that location using the integral image.

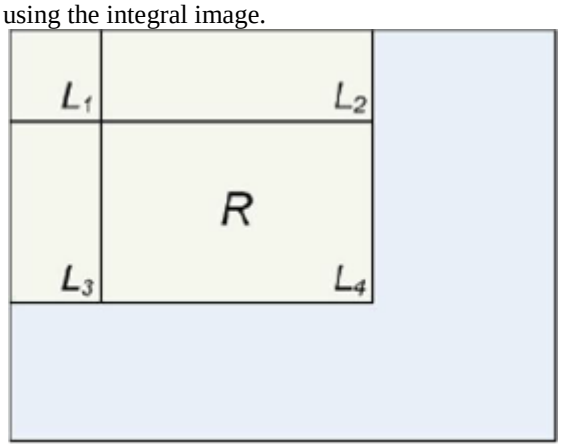

Figure 6: Calculating the area of a rectangle R is done using the corner of the rectangle: L4-L3-L2+L1.

By using each corner of a rectangle, the area can be computed quickly as denoted by Fig. 6. Since L1 is subtracted off twice it must be added back on to get the correct area of the rectangle. The area of the rectangle R, denoted as the rectangle integral, can be computed as follows using the locations of the integral image: L4-L3-L2+L1.

D. Cascade Classifier

The Viola and Jones face detection algorithm eliminates face candidates quickly using a cascade of stages. The cascade eliminates candidates by making stricter requirements in each stage with later stages being much more difficult for a candidate to pass. Candidates exit the cascade if they pass all stages or fail any stage. A face is detected if a candidate passes all stages. This process is shown in Fig. 7.

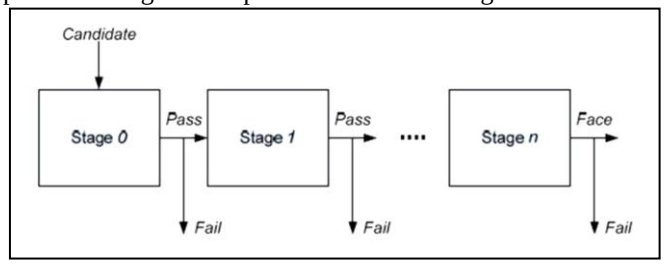

Figure 7: Cascade of stages. Candidate must pass all stages in the cascade to be concluded as a face.

E. Integral Image

The integral image is defined as the summation of the pixel values of the original image. The value at any location (x, y) of the integral image is the sum of the image's pixels above and to the left of location (x, y). Fig. 8 illustrates the integral image generation.

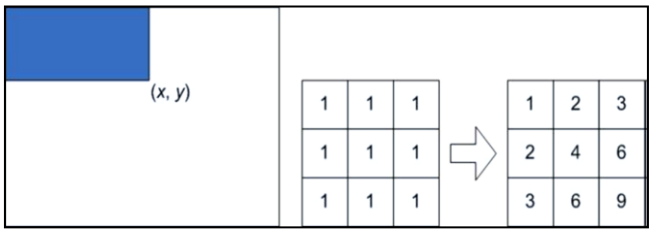

Figure 8: Integral image generation. The shaded region represents the sum of the pixels up to position (*x*, *y*) of the image. It shows a 3×3 image and its integral image representation.

F. Haar Feature Classifier

A Haar feature classifier uses the rectangle integral to calculate the value of a feature. The Haar feature classifier multiplies the weight of each rectangle by its area and the results are added together. Several Haar feature classifiers compose a stage. A stage comparator sums all the Haar feature classifier results in a stage and compares this summation with a stage threshold. The threshold is also a constant obtained from the AdaBoost algorithm. Each stage does not have a set number of Haar features. Depending on the parameters of the training data individual stages can have a varying number of Haar features. For example, Viola and Jones‟ data set used 2 features in the first stage and 10 in the second. All together they used a total of 38 stages and 6060 features [6]. Our data set is based on the OpenCV data set which used 22 stages and 2135 features in total.

The face detection algorithm proposed by Viola and Jones is used as the basis of our design. The face detection algorithm looks for specific Haar features of a human Upper-body. When one of these features is found, the algorithm allows the face candidate to pass to the next stage of detection. A face candidate is a rectangular section of the original image called a sub-window. Generally these sub-windows have a fixed size (typically 24×24 pixels). This sub-window is often scaled in order to obtain a variety of different size faces. The algorithm scans the entire image with this window and denotes each respective section a face candidate.

G. Region of Interest

A Region of Interest was selected based on subset of samples within a dataset identified for a particular purpose. The ROI is set to the coordinates after calculating the cumulative sum and the ROI is cropped and saved to the permanent memory. The region with rectangular shape inside an image is known as Region of Interest, this method is used to segment object for other processing.

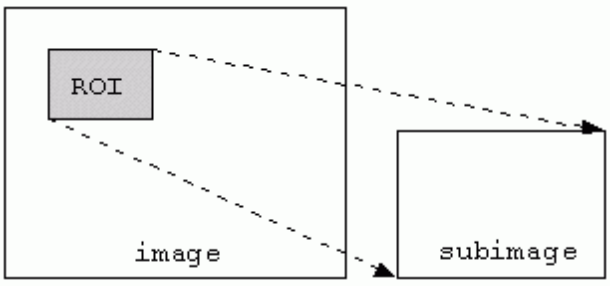

Figure 9: Image with Region of Interest (ROI)

The Fig. 9 shows the selection of sub-image Using ROI In Fig. 9, a Region of Interest is well-defined at upper leftward of image. When the ROI defined, many OpenCV tasks will accomplished on that specific place. This gives extra advantageous, if object has to crop from an image or to achieve pattern matching in sub-image. Always, Region of Interest should be within image. The Fig. 10 shows the cropping of image with OpenCV. The region of interest is given in the form of height, width and depth in the command window. For example 60:80:210 then the given area is being taken as the region of interest.

```
#Set Region of interest from the original image...
#Here cropping the image from minimum row index tothe max row of a cartoon strip
cvSetImageROI( original_image, cvRect(
min_col,min_row, (max_col-min_col),(max_rowmin_
row) ) );
#Create a cropped image from the source image
IplImage* cropped = cvCreateImage(
cvGetSize(original_image),original_image-
>depth,original_image->nChannels );
#Do the copy
cvCopy( original_image, cropped, NULL);
cvResetImageROI( original_image );
#Show cropped image
cvNamedWindow( "cropped",
```

```
CV_WINDOW_AUTOSIZE) ;
cvShowImage( "cropped", cropped );
cvWaitKey();
#Release the image window
cvReleaseImage(&cropped);
```

Figure 10: Algorithm for cropping with OpenCV

H. OpenCV Images Store

IplImage is a C structure to store OpenCV images,. Image Processing Library (IPL) is inheritance from the actual OpenCV versions, which may require. IplImage data type is defined in CXCORE. Furthermore to raw pixel data, which comprises many of imaginative fields, together called the Image Header. Which include **width** - width of the image in pixels and **height** – height of image in pixels, **depth** – is numerous predefined coefficients, which designate the number of bits per pixel per channel.

V. Experimental Results And Discussions

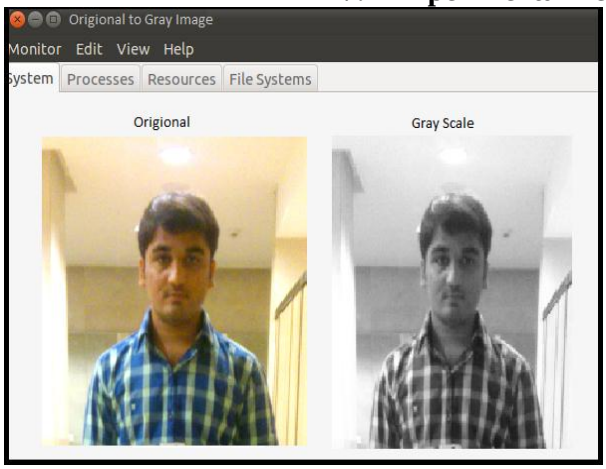

Figure 11: RGB to GreyScale image conversion

In this section the experimental results are presented. In first stage, the simulation was done using OpenCV 2.0.0.CMAKE 2.8 GUI with Qt creator. The image of color to grey scale conversion is shown in Fig. 11. After that Haar Window finds haar feature of this Gray scale image and make a .xml file which was stored at boot loader itself. Now incoming image frames from camera is also calculate haar feature and compare with predefined .xml file. Now these incoming image frames pass all stage of cascade classifier and indicate that, it is human upper body or not? If it detected than result as shown in figure 12 will be output

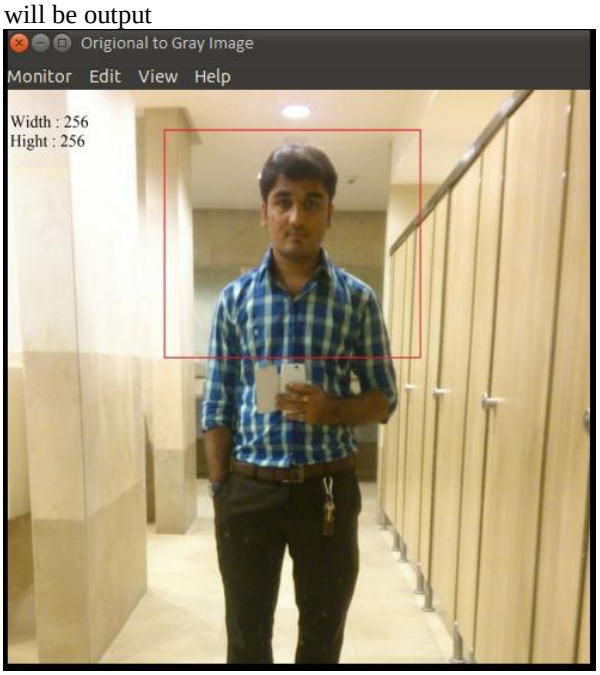

Figure 12: Detected upper Body Output

VI. Conclusions

In this paper, we presented a method to detect human upper body using Embedded Platform. ARM9 Mini 2440 board is used to compilation of OpenCV image processing algorithm. The developed software was able to slice or chop the comics strip along border into individual units, stored on the permanent memory and displayed on the ARM9 Board. Human detection algorithm works on the basis of Haar like Features and this haar features are classify with using cascade classifier which is addition of weak classifier makes a strong classifier. This developed software can be easily integrated in ARM9 Mini 2440 by the cross compilation of QT and the OpenCV software for an Operating system.

Acknowledgment

Here we are thanking to our collage to allow this research work and also to all Pre-Researchers

References

[1] Khana, The future of Mobile Application Storefronts, Wireless Expertise market report, 2009
[2] Fitzek and Reichert, Mobile Phone Programming and its Application to Wireless Networking, 2007
[3] Jurgen Scheible and Ville Tuulos, Mobile Python: Rapid
[4] prototyping of applications on the mobile platform, 2007
[5] http://sourcefouge.net/projects/opencvlibrary.
[6] Willow Garage, Inc., Willow garage http://www.willowgarage.com/ (August 2009).

[7] J.-P. Farrugia, P. Horain, E. Guehenneux, Y. Alusse, GPUCV: a framework for image processing acceleration with graphics processors, in: Proceedings of the ICME, 2006.
[8] Masayuki Hiromoto, Kentaro Nakahara, Hiroki Sugano,Yukihiro Nakamura, and Ryusuke Miyamoto, "Aspecialized processor suitable for adaboost-based detection with haar-like features," in Proc. of CVPR, 2007.
[9] Clemens Arth, Florian Limberger, and Horst Bischof,"Real-time license plate recognition on an embedded DSP-platform," in Proc. of CVPR, June 2007.
[10] S. Dikbas, T. Arici, and Y. A1tunbasak, "Chrominance Edge preserving Grayscale Transformation with Approximate First Principal Component for Color Edge Detection", IEEE Proc. ICIP 2007,11,261-264,2007.
[11] K. Panetta, S. Qasi, and S. Agaian, "Techniques for detection and classification of edges in color images", SPIE Proc., Volume 6982, pp. 69820W-69820W-11 (2008).
[12] K. Panetta, S. Qasi, and S. Agaian, "Techniques for detection and classification of edges in color images", SPIE Proc., Volume 6982, pp. 69820W-69820W-11 (2008).
[13] H.M. Merklinger, "A Technical View of Bokeh," Photo Techniques,1997
[14] Linear Gaussian blur evolution for detection of blurry images E. Tsomko1 H.J. Kim1 E. Izquierdo2
[15] X. Wang and J. Jian-Qiu, "An edge detection algorithm based on Canny operator". IEEE Proc. of the seventh International Conference on Intelligent Systems Design and Applications (ISDA2007),623-628, 2007
[16] E.J. Wharton, K. Panetta , and S. Agaian, "Logarithmic edge detection with applications"., Journal of computers, 3(9), 2008.
[17] S. Wesolkwoski and Ed. Jernigan, "Color Edge Detection IN RGB using Jointly Euclidean Distance and Vector Angle". University of Waterloo,Waterloo Canada, Vision Interface 9- 16, 1999
[18] A. Koschan, "A Comparative Study on Color Edge Detecton", IEEE Proc. ACCV'95, Singapore, IlL 547-478,1995
[19] S. Dikbas, T. Arici, and Y. A1tunbasak, " Chrominance Edge preserving Grayscale Transformation with Approximate First Principal Component for Color Edge Detection", IEEE Proc. ICIP 2007,11,261-264,2007.
[20] A. Koschan, "A Comparative Study on Color Edge Detecton", IEEE Proc. ACCV'95, Singapore, IlL 547- 478,1995.
[21] JOHN CANNY "A Computational Approach to Edge Detection" IN IEEE Transactions on Pattern Analysis and Machine Intelligence, 679-698, 1986.