REAL TIME FACE DETECTION AND TRACKING USING HAAR CLASSIFIER ON SOC

[1]R N DASCHOUDHARY, [2]RAJASHREE TRIPATHY

[1]Professor, [2]Research Scholar, [1,2]Department of EIE -Siksha O Anusandhan University

Abstract- In this paper we intend to Implement a real time Face detection and tracking the head poses position's from high definition video using Haar Classifier through Raspberry Pi BCM2835 CPU processor which is an combination of SoC with GPU based Architecture. OVA5647 CMOS Image sensor with 5-megapixel used for obtaining high definition video H.264 video data via GPU's hardware video decoder to improve playback of H.264 Video data supporting from 1080p at 30fps with complete user control over formatting and output data transfer also supporting with 720p/60HD video in full field of View(FOV). SimpleCV and OpenCV libraries are used for face detection and tracking the head poses position. The experimental result computed by using computer vision SimpleCV and OpenCV framework libraries along with above mentioned hardware results in a sustained throughput of 30 fps under 1080p resolutions for with higher accuracy and speediness for face detection and tracking the head poses position.

Index Terms- Computer Vision, Face Detection, Haar Classifier, Tracking

I. INTRODUCTION

Face detection from a image has been playing a vital role in the active research area, especially in computer vision research for more decades beginning from Eigen Faces, Principal Component Analysis, for numerous applications from monitoring, surveillance, Biometric and Human Computer Interaction. Face detection proposed by Viola and Jones's, Face detection is most used face detection is based on statistical methods for rapid frontal face detection system using Haar-like features. Analyzing the pixels for face detection is time consuming and difficult to accomplish because of the wide variations of shape and pigmentation within a human face.

Viola and Jones devised an algorithm, called Haar Classifiers, to rapidly detect any object, including human faces, using AdaBoost classifier cascades that are based on Haar-like features and not pixels. Haar –Like features widely used for identifying and locating human face in images regardless of size, position and condition including color, texture and motion under dynamic environments and the resources of the systems are considered to be monopolized by face detection.

In this paper we intend to implement the Haar-Classifier for Face detection and tracking based on the Haar-Features on System on Chip (SoC) for use in a human machine interface and action interpretation. Haar-like features can be computed at any scale or location in constant time using the integral image representation for images. The problem in analysis the pose is binary classification, which needs to be classified in every images and segregate data belonging to the class of interest (i.e. faces), or not. Most of the researchers are contributing focused to the two major main aspects of the problem: Haar-like feature and Classifier. Haar-Like features are used for selecting classes of features by improving the performance of the classifier based on the trained set of data pertaining to them. Haar Classifier adapts the existing classification algorithms to the detection and recognition problems.

II. RELATED WORKS

Robust and real-time face detection place a vital role in many of the application scenarios like access control, surveillance scenarios, gaming, human-computer interaction, etc. Viola and Jones devised an algorithm, called Haar Classifiers, to rapidly detect any object, including human faces, using Haar classifier cascades that are based on Haar –Like features. Haar-like features and not pixels. Different types of methods are available for detecting the face and recognition: Principal Component Analysis (PCA), Linear Discriminate Analysis (LDA), Support Vector Machines (SVM) Independent Component Analysis (ICA), Local Binary Pattern (LBP), and more recently Sparse Representation (SR) based methods. A recent survey on face recognition algorithms can be found in. Different algorithm are existing for performing and analysis of face detection with each of its own weakness and strengths related to use of flesh tones, some use contours, and other are even more complex involving templates, neural networks, or filters few of these algorithm are computationally expensive. There has been little work in the literature during the last years about real-time face detection at HDTV resolutions. Face detection algorithm using Haar-like features was described by Viola and Jones [14] and now it and a range of its modifications are widely spread in many applications. One of these modifications [15] was implemented in OpenCV library [16]. The OpenCV implementation compiled with OpenMP option provides only 4.5 frames per second on 4-core CPU. It's too slow to process HD stream in real time. As a solution to this problem a parallel modification of OpenCV algorithm for GPU has been developed.

Some parallel versions of face detection algorithm using Haar-like features [17, 18, 19]. The algorithm introduced by Hefenbrock [16] was the first realization of a face detection algorithm using GPU we could find.

It showed an effect of using GPU versus CPU. But the algorithm could not process a stream with resolution 640x480 in real time. The next parallel implementation is found in Obukhov's algorithm [20]. It's a single realization that uses GPU and can work with OpenCV classifiers without modification that is why modern versions of OpenCV library include it (test result of the algorithm is presented. in corresponding section of the paper). The main problem of the algorithm is texture memory usage for classifier storing because texture memory is not as effective for general operation as cached global memory on modern GPU.

Section III presents to the face detection mechanism that uses Haar Classifiers based on Haar-like features. Section II refers to related works. Section IV presents the face detection design and analysis with Cascade Haar Classifier tracking mechanism based on a Haar-Like features approach. Section V presents the System overview obtained by implementing the Haar Classifier on System on Chip (Raspberry Pi). Further section real time data results are presented where it can be seen that faces are detected in images from High Definition video streaming from Image Sensor. Section VI provides result and Section VII concludes this article.

III. FACE DETECTION HAAR CLASSIFIER ALGORITHM

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 face. 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 [6]. The algorithm uses an integral image in order to process Haar features of a face candidate in constant time. It uses a cascade of stages which is used to eliminate non-face candidates quickly. Each stage consists of many different Haar features. Each feature is classified by a Haar feature classifier. The Haar feature classifiers generate an output which can then be provided to the stage comparator. The stage comparator sums the outputs of the Haar feature classifiers and compares this value with a stage threshold to determine if the stage should be passed. If all stages are passed the face candidate is concluded to be a face. These terms will be discussed in more detail in the following sections.

A. 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). Figure 1 illustrates the integral image generation. The simple rectangular features of an image are calculated using an intermediate representation of an image, called the integral image [1]. The integral image is an array containing the sums of the pixels' intensity values located directly to the left of a pixel and directly above the pixel at location (x, y) inclusive. So if $A(x, y)$ is the original image and $AI[x, y]$ is the integral image then the integral image is computed as shown in equation 1 and illustrated in Figure 2.

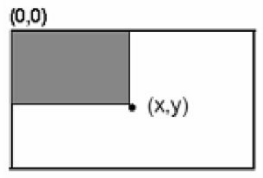


Figure 1. Summed Area of Integral Image

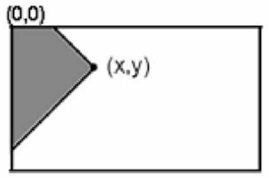


Figure 2. Summed Area of Rotated Integral Image

$$AI[x, y] = \sum_{x' \le x', y' \le y} A(x', y') \qquad (1)$$

The features rotated by forty-five degrees, like the line feature shown in Figure 1(b), require another intermediate representation called the rotated integral image or rotated sum auxiliary image[1]. The rotated integral image is calculated by finding the sum of the pixels' intensity values that are located at a forty five degree angle to the left and above for the x value and below for the y value. So if $A[x, y]$ is the original image and $AR[x,y]$ is the rotated integral image then the integral image is computed as shown in equation 2 an illustrated in Figure 2 and Figure 3.

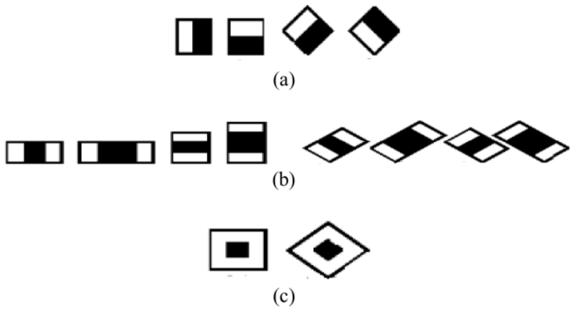


Figure 3. (a) Edge Features (b) Line Features (c) Center Surround Features

$$AR[x,y] = \sum_{x' \le x', x' \le x - |y - y'|} A(x', y') \qquad (2)$$

It only takes two passes to compute both integral image arrays, one for each array. Using the appropriate integral image and taking the difference between six to eight array elements forming two or three connected rectangles, a feature of any scale can be computed. Thus calculating a feature is extremely fast and efficient. It also means calculating features of various sizes requires the same effort as a feature of only two or three pixels. The detection of various sizes of the same object requires the same amount of effort and time as objects of similar sizes since scaling requires no additional effort.

B. Haar Features

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. The weights are constants generated by the learning algorithm. There are a variety of forms of features as seen below in Figure 4. 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. By using each corner of a rectangle, the area can be computed quickly as denoted by Figure 5. Since L 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:

$$L_4 - L_3 - L_2 + L_1$$

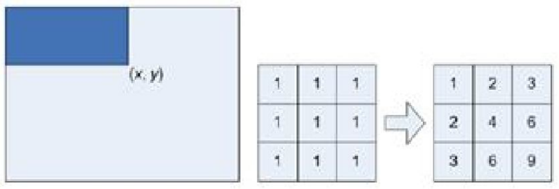


Figure 4. Integral Image generation. It shows a 3x3 image and its integral image representation

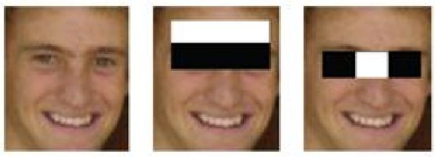


Figure 5. Examples of Haar Feature .Area of White and black region are multiplied by their respective weights and then summed in order to get the HAAR Feature Value

C. 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 as shown in the Figure 6. 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. Our data set is based on the OpenCV data set which used 22 stages and 2135 features in total.

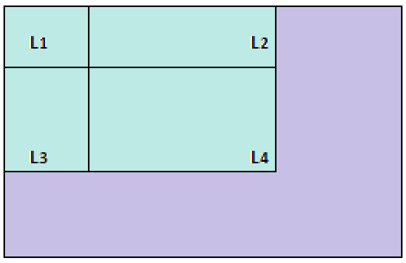


Figure 6. Calculating the area of rectangular

$$L_4 - L_3 - L_2 + L_1$$

D. Cascade

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 Figure 7.

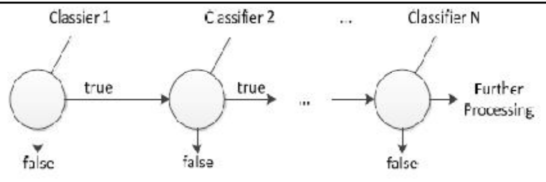


Figure 7. Cascaded of stages.Must pass all the stages to detect Face

IV. FACE DETECTION DESIGN AND ANALYSIS

This section will describe about the Face detection with itself which has several modules that are working together as one to make the system runs smoothly. The phase consists of capture image; Detect faces in the image, feature extraction, template comparison, declaration of matching template. The acquisition of face images can be done by acquiring the real time image from the OVA5647 CMOS Image Sensor interfaced with Raspberry pi High speed processor with GPU Processing. Furthermore the acquisition can also be done through real time remote monitoring either with IP or Wifi. The function of the face detection module is clarify whether the face is available during real time monitoring or detection of not. The face detection is done by scanning up an image for different scales and looking for some simple patterns as mentioned in the above section III. When the system detects the face, it will produce and sub-image that is scaled such that the face appears in the center and presented at a uniform size. OpenCV already provide algorithm to locate faces in still image and videos-Algorithm mentioned in section III. Haar classifier algorithm scans the image and creates a bounding box as returns for each detected face.

The feature extraction in face detection is done by localizing of the characteristics of face components (i.e., eyes, mouth, nose etc) in an image. In other terms the feature extraction is a step in face recognition where the system locates certain points on the faces such as corner and center of the eyes, tip of the nose, mouth etc. It analyzes spatial geometry of differential feature of a face. Result of this analyzing is a set of template generated for each face .The template consists of a reduced set of data which represent the real time face detected in bounded box. The template comparison is done with the template stored in the database. Two phases are there in this phase Identification and verification. These two term identification to detect the face in real time video and verification application for face recognition which scope out of this paper. The final phase of face detection is to declare the highest matching score resulted in the previous step.

The configuration will determine how the application should behave based on the desired security and operational consideration. The face detection methodology is shown in the Figure 8.

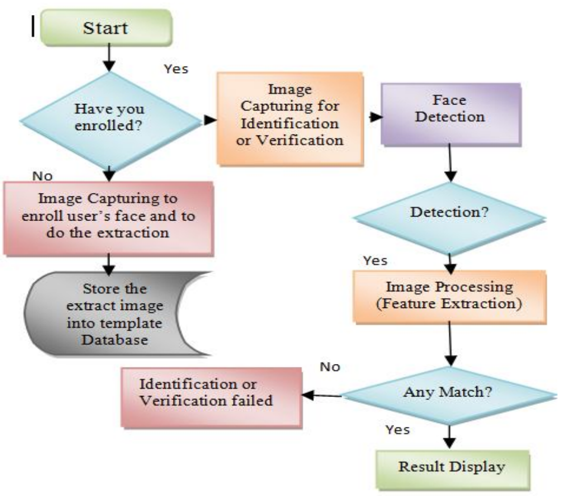

Start
Have you enrolled?
Yes
No
Image Capturing for Identification or Verification
Face Detection
Image Capturing to enroll user's face and to do the extraction
Detection?
Yes
Store the extract image into template Database
Image Processing (Feature Extraction)
No
Identification or Verification failed
Any Match?
Yes
Result Display

Figure 8. Flowchart of face detection system

A. Face detection

System is capable of detecting the faces from the captured image for the purpose of prototype. From the above Section, face detection determines where in an image a face is located and it is done by scanning the different image scales and extracting the exact patterns to detect the face. The Prototype is to built with Haar-Like Feature function from OpenCV. Haar classifier detection is used to create a search window that slide through a image and check whether a certain region of an image looks likes face or not. Haar like features and a large set of very weak classifier uses a single feature to define a certain image as face or non face. Each feature is described by the template its coordinate relative to the search window origin and the size of the feature.

The search window quickly scanning the first classifier on the cascade as shown in the Figure 9, if the classifier returns false then the computation on that window also ends and results no detected face(false).Moreover, if the classifier returns true, then the window will be passed down to the next classifier in the cascade to do the exact same thing. When all classifier return true for that window, then the result will returns true also for that certain window face detected.

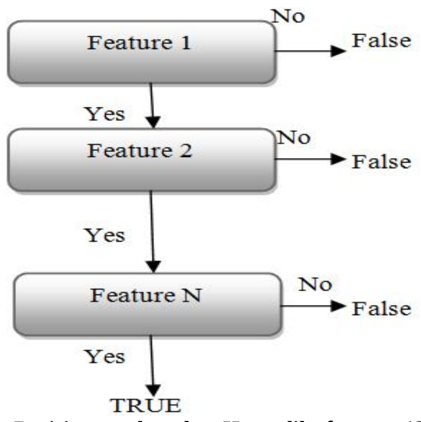


Figure 9. Decision tree based on Haar –like features (Cascade of classifier)

V. SYSTEM DESIGN

This paper is demonstrate and to utilize the portability of the Raspberry Pi by developing a security system using the camera module extension. The security system should have the capability of capturing a live video stream, detect any faces in the frame and recognize the face using a facial recognition algorithm. Feedback in the form of speech should be integrated into the security system shown in the below Figure 10 through remote monitoring for detecting the face interface with Raspberry Pi and OVA5647 CMOS Image Sensor which has High Definition Video processing.

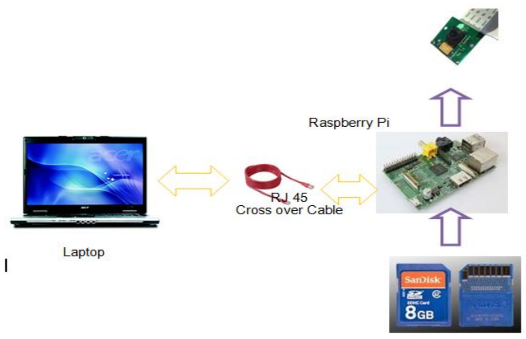


Figure 10. Remote monitoring Face Detection System Raspeberry pi with OVA5647

The image processing tasks should be completed using the OpenCV library developed by Intel, which is compatible with the Raspberry Pi board. Figure 11 Shows the real time face detection system.

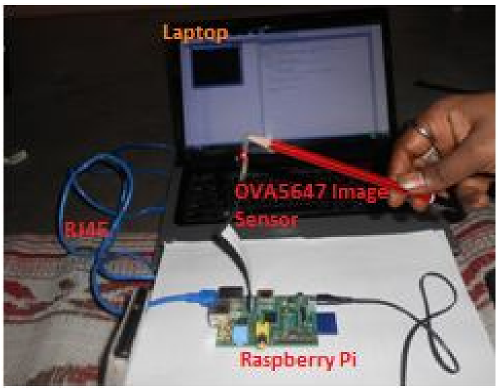
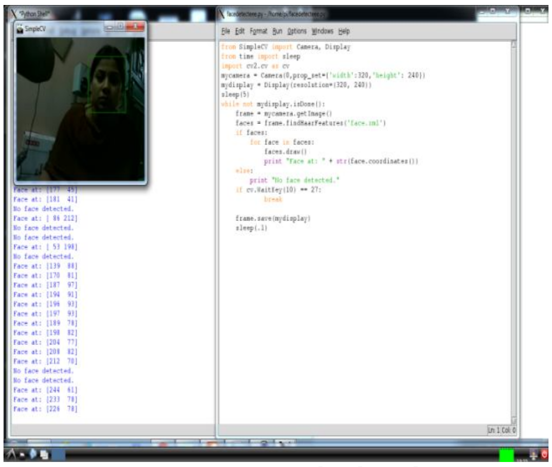


Figure 11. Real time Face Detection System

VI. RESULTS

The result of face detection is shown in Figure 12. Those are the frames extracted from the video. Sometimes, face detection algorithm may get more than one result even there is only one face in the frame, such as Figure 12. In this case, the post processing has been used. If the detector provides more than one rectangle, which indicates the position of the face, the distance of center points of these rectangles has been calculated. If this distance is smaller than a pre-set threshold, the average of these rectangles will be computed and set as the final position of the detected face.

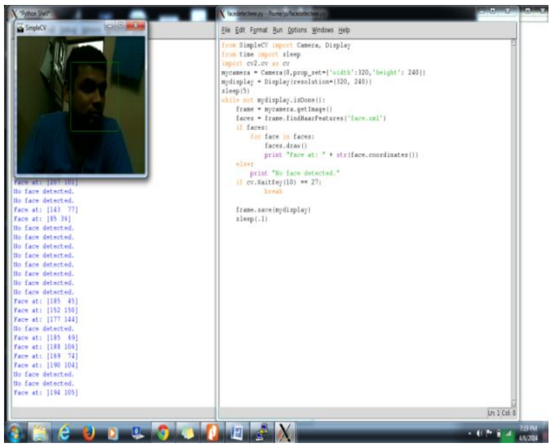
In the paper we implement face tracking in the Python by using Viola Jones face detection. This method is verified and the limitations of the scheme are observed through testing and debugging our codes. And then, limited by Python performance, we shift to OpenCV to evaluate the speed of this face tracking scheme, We found the Viola Jones face detection is more suitable for real-time face detection since they requires less CPU resource and costs shorter time.


"Python Shell"
SimpleCV
facedetecteee.py - /home/pi/facedetecteee.py
File Edit Format Run Options Windows Help
Ln: 1 Col: 0

```
from SimpleCV import Camera, Display
from time import sleep
import cv2.cv as cv
mycamera = Camera(0,prop_set={'width':320,'height': 240})
mydisplay = Display(resolution=(320, 240))
sleep(5)
while not mydisplay.isDone():
    frame = mycamera.getImage()
    faces = frame.findHaarFeatures('face.xml')
    if faces:
        for face in faces:
            faces.draw()
            print "Face at: " + str(face.coordinates())
    else:
        print "No face detected."
    if cv.WaitKey(10) == 27:
            break

    frame.save(mydisplay)
    sleep(.1)
```

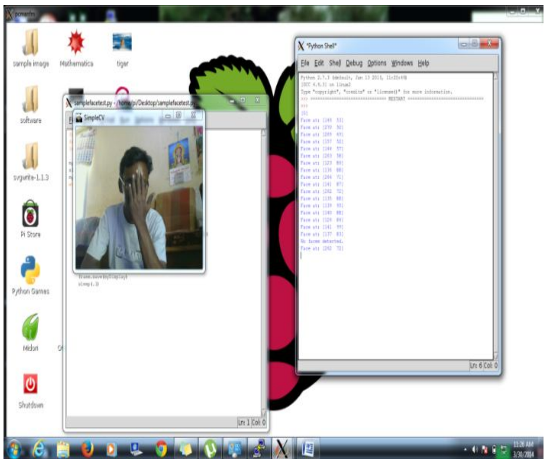
```
Face at: [177  45]
Face at: [181  41]
No face detected.
Face at: [ 86 212]
No face detected.
No face detected.
Face at: [ 53 198]
No face detected.
Face at: [139  88]
Face at: [170  81]
Face at: [187  97]
Face at: [194  91]
Face at: [196  93]
Face at: [197  93]
Face at: [189  78]
Face at: [198  82]
Face at: [204  77]
Face at: [208  82]
Face at: [212  70]
No face detected.
No face detected.
Face at: [244  61]
Face at: [233  78]
Face at: [226  78]
```

Figure 12. Face Detected and Tracking



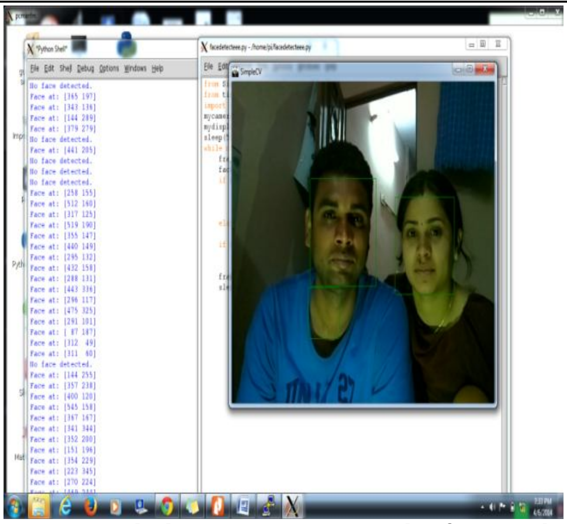
```
from SimpleCV import Camera, Display
from time import sleep
import cv2.cv as cv
mycamera = Camera(0,prop_set={'width':320,'height': 240})
mydisplay = Display(resolution=(320, 240))
sleep(5)
while not mydisplay.isDone():
    frame = mycamera.getImage()
    faces = frame.findHaarFeatures('face.xml')
    if faces:
        for face in faces:
            faces.draw()
            print "Face at: " + str(face.coordinates())
    else:
        print "No face detected."
    if cv.WaitKey(10) == 27:
            break

    frame.save(mydisplay)
    sleep(.1)
```

```
Face at: [207 101]
No face detected.
No face detected.
Face at: [143  77]
Face at: [85 36]
No face detected.
No face detected.
No face detected.
No face detected.
No face detected.
No face detected.
No face detected.
No face detected.
Face at: [185  45]
Face at: [152 150]
Face at: [177 144]
No face detected.
Face at: [185  69]
Face at: [188 106]
Face at: [169  74]
Face at: [190 104]
No face detected.
No face detected.
Face at: [194 105]
```



```
Python 2.7.3 (default, Jan 13 2013, 11:20:46)
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
[illegible]
Face at: [illegible]
No faces detected.
Face at: [illegible]
```

Figure 13. Face Not Detected



```
No face detected.
Face at: [365 197]
Face at: [343 136]
Face at: [144 289]
Face at: [379 279]
No face detected.
Face at: [441 205]
No face detected.
No face detected.
No face detected.
Face at: [258 155]
Face at: [512 160]
Face at: [317 125]
Face at: [519 190]
Face at: [355 147]
Face at: [440 149]
Face at: [295 132]
Face at: [432 158]
Face at: [288 131]
Face at: [443 336]
Face at: [296 117]
Face at: [475 325]
Face at: [291 101]
Face at: [ 87 187]
Face at: [312  49]
Face at: [311  60]
No face detected.
Face at: [144 255]
Face at: [357 238]
Face at: [400 120]
Face at: [545 158]
Face at: [367 167]
Face at: [341 344]
Face at: [352 200]
Face at: [151 196]
Face at: [354 229]
Face at: [223 345]
Face at: [270 224]
```

Figure 14. Multiple Face Detection –Haar Classifier Cascade

Table -1 Experiment Result

	Image 1	Image 2	Image3
Face Detection/Not	Face Detected	Face Not Detected	Multiple Face Detection
Position of the Face	[135 88] [262 71] [123 86]	[137 83]	[53 86] [173 110] [261 110]

Table 1 shows the detection and non detection of faces. The face detection is done by means of Cascade Haar Classifier. Continuous tracking of the face is done with Cascade classifiers for detecting the real time face detection. The result are obtained from Figure 12, Figure 13 and Figure 14.From Image 1 the face is detected and the tracking is observed from face positions varying from [135 88], [262 71], [123 86].Even multiple face detection is also possible with High Definition Video input.

CONCLUSION

Face detection and tracking is being challenging for many researchers with real time Image sensor. With the advancement the real time face detection in remote monitoring is help for building much efficient application. Moreover such technology can be useful in tracking the lost object under dynamic environment. Further enhancement of this work can be extended with stereo depth analysis of face detection using two image sensor interfaced with High speed Processor.

REFERENCES

[1] P. Viola and M. Jones, “Rapid object detection using a boosted classifier of simple features,” in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2001, vol. 1, pp. 511–518.

[2] P. Viola and M. Jones. Robust Real-time Object Detection. International Journal of Computer Vision, 57(2):137–154,2002. 2, 4

[3] Viola, P. and Jones, M. Rapid object detection using boosted cascade of simple features. IEEE Conference on Computer Vision and Pattern Recognition, 2001.

[4] M. Turk and A. Pentland, “Eigen faces for recognition,” Journal of Cognitive Neuroscience, vol. 3, no. 1, pp. 71– 86, 1991.

[5] K. Etemad and R. Chellappa, “Discriminant analysis for recognition of human face images,” Journal of the Optical Society of America A, vol. 14, no. 8, pp. 1724– 1733, 1997.

[6] B. Heisele, P. Ho, and T. Poggio, “Face recognition with support vector machines: global versus component based approach,” in Proceedings of IEEE International Conference on Computer Vision, 2001, vol. 2, pp. 688– 694.

[7] M.S. Bartlett, J.R. Movellan, and T.J. Sejnowski, “Face recognition by independent component analysis,” IEEE Transactions on Neural Networks, vol. 13, no. 6, pp. 1450–1466, 2002.

[8] T. Ahonen, A. Hadid, and M. Pietikainen, “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.

[9] J. Wright, A.Y. Yang, A. Ganesh, S.S. Sastry, and Y. Ma, “Robust face recognition via sparse representation,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 2, pp. 210–227, 2008.

[10] L. Ma, C. Wang, B. Xiao, and W. Zhou, “Sparse representation for face recognition based on discriminative low-rank dictionary learning,” in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2012, pp. 2586–2593.

[11] R. Chellappa, J. Ni, and V.M. Patel, “Remote identification of faces: Problems, prospects, and progress,” Pattern Recognition Letters, vol. 33, no. 14, pp. 1849– 1859, 2012.

[12] P. Viola, “Rapid object detection using a boosted cascade of simple features,” Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'01), vol.1, pp.511-518, 2001.

[13] R. Lienhart, “An extended set of Haar-like features for rapid object detection”, Proceedings of IEEE International Conference on Image Processing(ICIP'02), vol.1, pp.900-903, 2002.

[14] Open Computer Vision Library [Electronic resource], 2012, Mode of access: http://sourceforge.net/projects/opencvlibrary. – Date of access: 25.06.2012.

[15] J .Owens, “GPU architecture overview,” International Conference on Computer

[16] Graphics and Interactive Techniques (SIGRAPH’07), 2007.

[17] Whitepaper NVIDIA’s Next Generation CUDA Compute Architecture: Kepler GK110 [Electronic resource], NVIDIA, 2012, http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf.

[18] Daniel Hefenbrock, “Accelerating Viola-Jones face detection to FPGA-level using GPUs,” Proceedings of the 2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, 2010, pp.11-18.

[19] Anton Obukhov, “Haar Classifiers for Object Detection with CUDA”, GPU Computing Gems. Emerald Edition, 2011, pp.517-544.

[20] Adam Herout, “Real-time object detection on CUDA,” Journal of Real-Time Image Processing, vol.6, issue 3, 2011, pp.159-170.

[21] J. U. Duncombe, “Infrared navigation—Part I: An assessment of feasibility,” IEEE Trans. Electron Devices, vol. ED-11, pp. 34-39, Jan. 1959.

[22] C. Y. Lin, M. Wu, J. A. Bloom, I. J. Cox, and M. Miller, “Rotation, scale, and translation resilient public watermarking for images,” IEEE Trans. Image Process., vol. 10, no. 5, pp. 767-782, May 2001.

★★★