**Colour Independent background and foreground Text Extraction using Python OpenCV on Stock Images data base**

**PBL REPORT**

**In**

**Image and Vision Computing**

**Submitted for the degree of**

**Bachelor of Technology**

***by***

**Marla Akhil Reddy (13BCE1078)**

**Chava Sai Teja (13BCE1032)**

**Sailesh Patra (13BCE1119)**

**Under the Guidance**

**Dr.G.Malathi**

**Associate Professor**

**School of Computing Science and Engineering**



**VIT University**

**Chennai- 600 127 – INDIA**

**November 2015**

**CONTENTS**

|  |  |
| --- | --- |
| **ABSTRACT** | **3** |
| **Introduction** | **4** |
| **Scope of the Project** | **7** |
| **Software/ Tools used** | **8** |
| **Dataset** | **9** |
| **Methodology** | **15** |
| **Implementation** | **17** |
| **Coding** | **22** |
| **Screen shots** | **39** |
| **Conclusion** | **60** |
| **References** | **61** |
| **Acknowledgement** | **62** |

**ABSTRACT**

**In recent times, we see lot more Informative and advertising mediums like flexes, covers and digital images having multi color and texture background, making it difficult to extract exact information .It’s difficult to use various technologies like OCR, which have high value, on this type of digital images.**

**To solve this issue we can use various binerization and extraction techniques to use in colour documents whereby the foreground text is output as black and the background as white regardless of the polarity of foreground-background shades.**

**All the available methods can sperate only text but our proposed method can completely separate text background and foreground ground to two different colours namely white and black respectively.**

**So we can use a novel method for binarization of color documents whereby the foreground text is output as black and the background as white regardless of the polarity of foreground-background shades. The method employs an edge-based connected component approach and automatically determines a threshold for each component. It has several advantages over existing binarization methods. Firstly, it can handle documents with multi-colored texts with different background shades. Secondly, the method is applicable to documents having text of widely varying sizes, usually not handled by local binarization methods. Thirdly, the method automatically computes the threshold for binarization and the logic for inverting the output from the image data and does not require any input parameter. The proposed method has been applied to a broad domain of target document types and environment and is found to have a good adaptability.**

**INTRODUCTION**

**The use of cameras has greatly eased document acquisition and has enabled human interaction with any type of document. Its ability to capture non-paper document images like scene text has several potential applications like licence plate recognition, road sign recog- nition, digital note taking, document archiving and wearable computing. But at the same time, it has also presented us with much more challenging images for any recognition task. Traditional scanner-based document analysis systems fail against this new and promising acquisition mode. Camera images suffer from uneven lighting, low resolution, blur,** **and perspective distortion. Overcoming these challenges will help us effortlessly acquire and manage information in documents.**

**The simplest and earliest method is the global thresholding technique that uses a single threshold to classify image pixels into foreground or background classes. Global thresholding techniques are generally based on histogram analysis. It works well for images with well separated foreground and background intensities. However,most of the document images do not meet this condition and hence the application of global thresholding methods is limited. Camera-captured images often exhibit non-uniform brightness because it is difﬁcult to control the imaging environment unlike the case of the scanner. The histogram of such images is generally not bi-modal and a single threshold can never yield an accurate binary document image. As such, global binarization methods are not suitable for camera images. On the other hand, local methods use a dynamic threshold across the image according to the local information. These approaches are generally window-based and the local threshold for a pixel is computed from the gray values of the pixels within a window centred at that particular pixel.**

**Niblack proposed a binarization scheme where the threshold is derived from the local image statistics. The sample mean μ(x,y) and the standard deviation σ(x,y) within a window W centred at the pixel location (x,y) are used to compute the threshold T(x,y) as follows:**

**T(x,y) = μ(x,y) − k σ(x,y), k = 0 . 2**

**Yanowitz and Bruckstein [10] introduced a threshold that varies over different image regions so as to ﬁt the spatially changing background and lighting conditions. Based on the observation that the location and gray level values at the edge points of the image are good choices for local thresholds, a threshold surface is created by relaxation initialized on the edge points. The method is however computationally very intensive. Trier and Jain evaluated 11 popular local thresholding methods on scanned documents and reported that Niblack’s method performs the best for optical character recognition (OCR). The method works well if the window encloses at least 1-2 characters. However, in homogeneous regions larger than size of the window, the method produces a noisy output since the expected sample variance becomes the background noise variance. Sauvola and Pietikainen proposed an improved version of the Niblack’s method by introducing a hypothesis that the gray**

**Values of the text are close to 0 (Black) while the background pixels are close to 255 (White). The threshold is computed with the dynamic range of standard deviation (R) which has the effect of amplifying the contribution of standard deviation in an adaptive manner.**

**T(x,y) = μ(x,y) [1 + k (σ(x,y)R− 1)]**

**Where the parameters R and k are set to 128 and 0.5 respectively. This method minimizes the effect of background noise and is more suitable for document images. As pointed out by Wolf et al in [9], the Sauvola method fails for images where the assumed hypothesis is not met and accordingly, they proposed an improved threshold estimate by taking the local contrast measure into account.**

**T(x,y) = (1 − a ) μ(x,y) + aM + aσ(x,y)**

**Smax( μ(x,y) − M ) where M is the minimum value of the grey levels of the whole image, Smax is the maximum value of the standard deviations of all windows of the image and ‘a’ is a parameter ﬁxed at 0.5. The Wolf’s method requires two passes since one of the threshold decision parameter Smax is the maximum of all standard deviation of all windows of the images. The computational complexity is therefore slightly higher in this case. This method combines Savoula’s robustness with respect to background textures and the segmentation quality of Niblack’s method.**

**Documents with both graphics and text, where the text varies in color and size, call for more specialized binarization techniques. It is relatively difﬁcult to obtain satisfactory binarization with various kinds of document images. The choice of window size in local methods can severely affect the result of binarization and may give rise to broken characters and voids, if the characters are thicker than the size of the window considered. Moreover, we often encounter text of different colors in a document image. Conventional methods assume that the polarity of the foreground-background intensity is known a priori. If the polarity of the foreground background intensity is not known, the binary decision logic could treat some text as background and no further processing can be done on that text. Clark and Mirmhedi use simple decision logic to invert the result of binarization based on the assumption that the background pixels far out number the text pixels. Within each window, the numbers of pixels having intensity values higher or lower than the threshold are counted and the one which is less in number is treated as the foreground text. This simple inversion logic cannot handle the case where the characters are thick and occupy a signiﬁcant area of the window under consideration. Moreover, a document image can have two or more different shades of text with different background colors. Binarization using a single threshold on such images, without a priori information of the polarity of foreground-background intensities, will lead to loss of textual information as some of the text may be assigned as background. The characters once lost cannot be retrieved back and are not available for further processing. Possible solutions need to be sought to overcome this drawback so that any type of document could be properly binarized without the loss of textual information.**

**SCOPE OF THE PROJECT**

**Important aspect of this project is to get clear distinction between background and foreground for Text in digital Images.**

**So by using this method we can achieve more efficiency in field of image data extraction like OCR etc.. And we can customise Images depending on need like providing processed image to Aged people than Original multicolour Image.**

**On further developments, we can regenerate lost text by using modern AI techniques to reconstruct worn out documnts.**

**The edge detection method is good in ﬁnding the character boundaries irrespective of the foreground-background polarity. However, if the background is textured, the edge components may not be detected correctly due to edges from the background and our edge-box ﬁltering strategy fails.Overcoming these challenges is considered as a future extension to this work. The method is able to capture all the text while at the same time, ﬁlter out most of the components due to the background. The method can be extended to incorporate text localization as well.**

**Software/ Tools used**

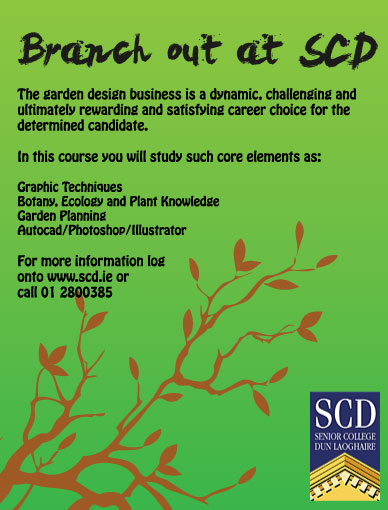
* **Python 3**
* **OpenCV Module to Python**
* **NumPy Module to Python**
* **Tesseract-OCR module**
* **OpenALPR to Python**
* **Automatic Image binerization tools for testing**
* **Edge detection module from github**

**DATASET**

**We took different colour images of different text sizes and with multiple colour background and texture to analyse the algorithum.**

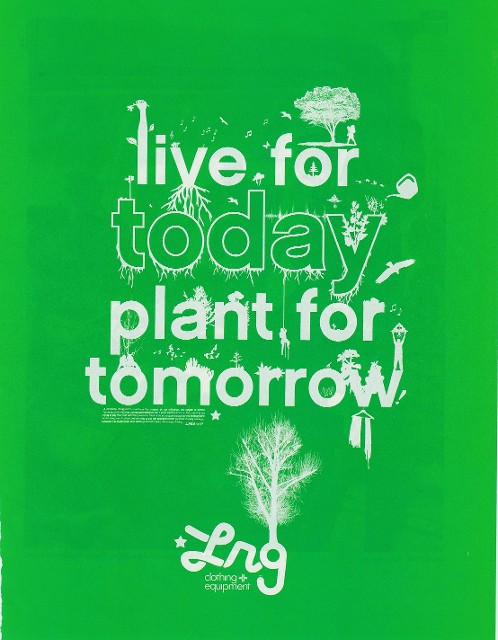
**So different images taken as Input Dataset are:**

****

****

****

****

**** ****

****

****

****

**METHODOLOGY**

**Text is the most important information in a document.We proposes a novel method to binarize camera captured color document images, whereby the foreground text is output as black and the background as white irrespective of the original polarity of foreground-background shades. The proposed method uses an edge-based connected component approach to automatically obtain a threshold for each component. Canny edge detection is performed individually on each channel of the color image and the edge map E is obtained by combining the three edge images as follows**

**E = ER ∨ EG ∨ EB**

**Here, ER, EG and EB are the edge images corresponding to the three color channels and ∨ denotes the logical OR operation. An 8-connected component labeling follows the edge detection step and the associated bounding box information is computed. We call each component, thus obtained, an edge-box (EB). We make some sensible assumptions about the document and use the area and the aspect ratios of the EBs to ﬁlter out the obvious non-text regions. The aspect ratio is constrained to lie between 0.1 and 10 to eliminate highly elongated regions. The size of the EB should be greater than 15 pixels but smaller than 1/5th of the image dimension to be considered for further processing.**

**Since the edge detection captures both the inner and outer boundaries of the characters, it is possible that an EB may completely enclose one or more Ebs. For example, the letter ‘O’ gives rise to two components; one due to the inner boundary EBint and the other due to the outer boundary EBout. If a particular EB has exactly one or two EBs that lie completely inside it, the internal EBs can be conveniently ignored as it corresponds to the inner boundaries of the text characters. On the other hand, if it completely encloses three or more EBs, only the internal EBs are retained while the outer EB is removed as such a component does not represent a text character. Thus, the unwanted components are ﬁltered out by subjecting each edge component to the following constraints:**

**If (Nint < 3)**

**{Reject EBint, Accept EBout}**

**Else**

**{Reject EBout, Accept EBint }**

**Where EBint denotes the EBs that lie completely in- side the current EB under consideration and Nint is the number of EBint. These constraints on the edge components effectively remove the obvious non-text elements while retaining all the text-like elements. Only the ﬁltered set of EBs are considered for binarization.**

**Estimation of Threshold**

**For each EB, we estimate the foreground and background intensities and the threshold is computed individually.The foreground and the background pixels which are used for obtaining the threshold and inversion of the binary output.**

**The foreground intensity is computed as the mean graylevel intensity of the pixels that correspond to the edge pixels.**

**For obtaining the background intensity, we consider three pixels each at the periphery of the corners of the bounding box as follows**

**B = {I(x − 1 , y − 1) , I( x − 1 , y ) , I( x, y − 1) ,I( x + w + 1 , y − 1) , I( x + w, y − 1) , I( x + w + 1 , y ) ,I( x − 1 , y + h + 1) , I( x − 1 , y + h ) , I( x, y + h + 1) ,I( x + w + 1 , y + h + 1) , I( x + w, y + h + 1) , I( x +w + 1 , y + h )}**

**Where (x, y) represent the coordinates of the top-left corner of the bounding-box of each edge component and w and h are its width and height, respectively. The output of the edge-box ﬁltering and the threshold parameters for each of the valid edge components. The mean or median intensity are almost the same for the foreground pixels irrespective of the text orientation. However, for a diagonally aligned text, the bounding boxes can have some overlap with the adjacent components and can interfere in the background intensity estimate. This is the case for the text ‘FLEXIBLE 6 CHANNEL’ printed in black in a semi-circular manner by the edge components whose estimated foreground intensity is lower than that of the background. The mean background intensity for these components are affected by the adjacent components while the median is not. Thus, the local background intensity can be estimated more reliably by considering the median intensity of the 12 background pixels instead of the mean intensity.**

**BEB = median (B)**

**Assuming that each character is of uniform color, we binarize each edge component using the estimated foreground intensity as the threshold. Depending on whether the foreground intensity is higher or lower than that of the background, each binarized output BWEB is suitably inverted so that the foreground text is always black and the background always white.**

**All the threshold parameters explained in this section are derived from the image data and the method is thus completely free from user-deﬁned parameters.**

**Implementation**

**Step 0:**

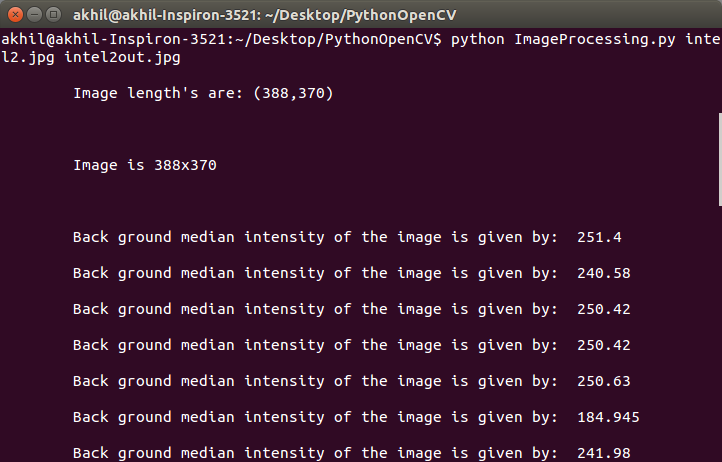
**We need to select a colour Image to be prossed.So we take below Image:**

****

**Step 1:**

**We need to open Python Intrepreter or terminal to run opencv module.**

**Now we need to run our PythonOpencv code as shown below:**

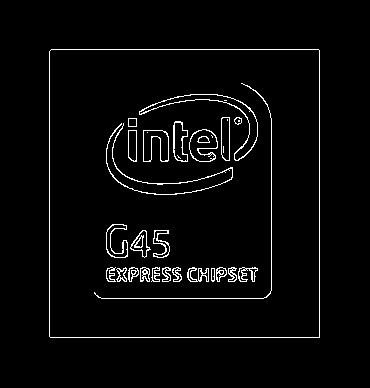
****

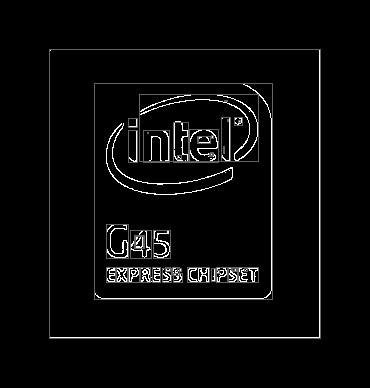
**We get Output Image as:**

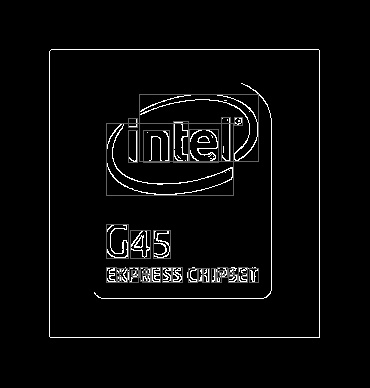
****

**Step 2:**

**Now we get different Images each specifiying Edges of Images, Rejected Boxes and Processed Boxes Respectively.**

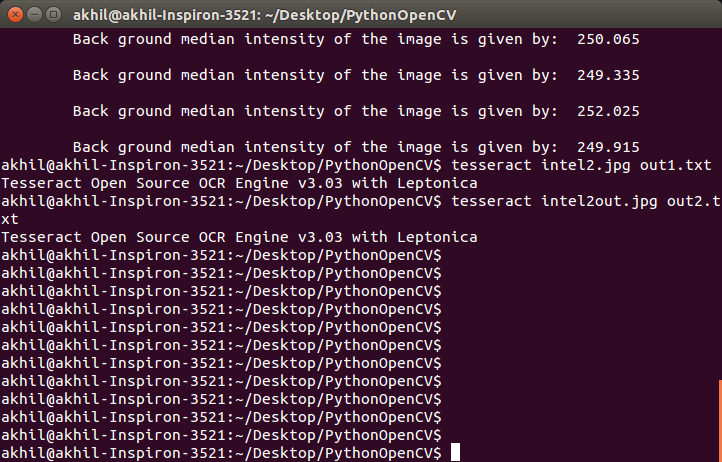
****

****

****

**Step 3:**

**Now we need to use tesseract (OCR Software) to get OCR of Input colour Image as shown:**

****

**Step 4:**

**Now we get OCR for both Input Image and Output Image as shown below:**

**OCR for Input Image:**

G45

EXPRESS CHIPSET

**OCR for Output Image:**

Lirltgl

G45

EXPRESS CNIPSET

**CODING**

'''

Project Code for :

"Colour Independent background and foreground Text Extraction using Python OpenCV on Stock Images data base"

\_User Name = Marla Akhil Reddy

\_Licensed to @M\_Akhil\_Reddy

'''

import cv2

import numpy as np

from matplotlib import pyplot as plt

import sys

import os.path

# Terminal AREA

#It's just to have proper Input format from terminal

if len(sys.argv) != 3:

print "Give Input file in Proper order:"

print "%s input\_file output\_file" % (sys.argv[0])

sys.exit()

else:

input\_file = sys.argv[1]

output\_file = sys.argv[2]

if not os.path.isfile(input\_file):

print "Sorry!No such file '%s'" % input\_file

sys.exit()

SET = 1

SET1 = 0

################################################################# Functions AREA ############################################################

def Intensity(xx, yy): # In this function, I am calculating pixel Intensity using formula ,pixel intensity = 0.30R + 0.59G + 0.11B

global img, img\_y, img\_x

if yy >= img\_y or xx >= img\_x: # It is out of bound condition for pixels

return 0

pixel = img[yy][xx]

return 0.30 \* pixel[2] + 0.59 \* pixel[1] + 0.11 \* pixel[0]

def connected(contour): # It's a quick function to test whether the contour is a connected shape or not

first = contour[0][0]

last = contour[len(contour) - 1][0]

return abs(first[0] - last[0]) <= 1 and abs(first[1] - last[1]) <= 1

def c(index): # Just a Helper function to return a given contour in required format

global contours

return contours[index]

'''

This function is just to Count the total number of real children

'''

def count\_children(index, hie, contour):

if hie[index][2] < 0: # No children

return 0

else:

if keep(c(hie[index][2])): #If the first child is a contour then we care about it and then count it, otherwise don't consider it

count = 1

else:

count = 0

count += count\_siblings(hie[index][2], hie, contour, True) #Finally count all of the child's siblings and their children too...

return count

def is\_child(index, hie): # Making a Quick check to test if the contour is a child

return get\_parent(index, hie) > 0

def get\_parent(index, hie): # Here we are Getting the first parent of the contour that we care about

parent = hie[index][3]

while not keep(c(parent)) and parent > 0:

parent = hie[parent][3]

return parent

'''

This function can be left as we are simply Counting the number of relevant siblings of a contour

'''

def count\_siblings(index, hie, contour, inc\_children=False):

if inc\_children: # Include the children if necessary

count = count\_children(index, hie, contour)

else:

count = 0

par = hie[index][0] # For Looking ahead of Child

while par > 0:

if keep(c(par)):

count += 1

if inc\_children:

count += count\_children(par, hie, contour)

par = hie[par][0]

n = hie[index][1] # For Looking behind of Child

while n > 0:

if keep(c(n)):

count += 1

if inc\_children:

count += count\_children(n, hie, contour)

n = hie[n][1]

return count

def keep(contour): # Check condition to whether we care about this contour or not ..

return KeepBox(contour) and connected(contour)

'''

Below function is to Whether we should keep the containing box of this contour based on it's shape or not

firstly, We need Test it's shape - if it's too a (rectangular object or flat figure with unequal adjacent sides) or tall .Then it's not a real character

Secondly, We need to check size of the box

'''

def KeepBox(contour):

xx, yy, w\_, h\_ = cv2.boundingRect(contour)

w\_ \*= 1.0

h\_ \*= 1.0

if w\_ / h\_ < 0.1 or w\_ / h\_ > 10:

if SET1:

print "\t Rejected because of shape: (" + str(xx) + "," + str(yy) + "," + str(w\_) + "," + str(h\_) + ")" + \

str(w\_ / h\_)

return False

if ((w\_ \* h\_) > ((img\_x \* img\_y) / 5)) or ((w\_ \* h\_) < 15): # This condition is for box

if SET1:

print "\t Rejected because of size"

return False

return True

'''

This function is used to get different child counters where boxes are included

'''

#global cse = 0

def include\_box(index, hie, contour):

cse = 0

cse = cse + 1

'''

if cse == 1:

print index

#print index,hie,contour

print "\t"

print hie

'''

if SET1:

print str(index) + ":"

if is\_child(index, hie):

print "\tIs a child"

print "\tparent " + str(get\_parent(index, hie)) + " has " + str(

count\_children(get\_parent(index, hie), hie, contour)) + " children"

print "\thas " + str(count\_children(index, hie, contour)) + " children"

if is\_child(index, hie) and count\_children(get\_parent(index, hie), hie, contour) <= 2:

if SET1:

print "\t skipping: is an interior to a letter"

return False

if count\_children(index, hie, contour) > 2:

if SET1:

print "\t skipping, is a container of letters"

return False

if SET1:

print "\t keeping"

return True

# ########################################################### Code AREA #####################################################################

orig\_img = cv2.imread(input\_file) # Taking Input file to object called orig\_img

'''

cv2.imshow('Akhil',orig\_img) # Here I am displaying Image being taken as Input

cv2.waitKey(5000)

cv2.destroyAllWindows()

'''

img = cv2.copyMakeBorder(orig\_img, 50, 50, 50, 50, cv2.BORDER\_CONSTANT) # Here I am adding border to make processing simple later and It's a

# a constant colored border and Window to varable object 'img'

# width and height of the image

img\_y = len(img)

img\_x = len(img[0])

print "\n\tImage length's are: (%d,%d) " % (img\_y,img\_x)

if SET:

print "\n\t"

print "\n\tImage is " + str(len(img)) + "x" + str(len(img[0]))

print "\n\t"

blue, green, red = cv2.split(img) #Splitting Image into each channel seperately

# Here we are Running canny edge detection on each channel

blue\_edges = cv2.Canny(blue, 200, 250)

green\_edges = cv2.Canny(green, 200, 250)

red\_edges = cv2.Canny(red, 200, 250)

'''

# Using Canny Edge Detection to show edges of image

edges = cv2.Canny(img,100,200)

plt.subplot(121),plt.imshow(img,cmap = 'gray')

plt.title('Original Image'), plt.xticks([]), plt.yticks([])

plt.subplot(122),plt.imshow(edges,cmap = 'gray')

plt.title('Edge Image'), plt.xticks([]), plt.yticks([])

plt.show()

'''

edges = blue\_edges | green\_edges | red\_edges # Joining edges back into image

contours, hierarchy = cv2.findContours(edges.copy(), cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE) # Find the contours to the image and getting into

# proper hierarchy

'''

Using: Douglas-Peucker algorithm

Three arguments in cv2.findContours() function, first one is source image, second is contour retrieval

mode, third is contour approximation method. And it outputs the image, contours and hierarchy. contours is a

Python list of all the contours in the image. Each individual contour is a Numpy array of (x,y) coordinates of boundary

points of the object.

'''

#print hierarchy

#print hierarchy[0]

#print hierarchy[1]

hierarchy = hierarchy[0]

if SET:

processed = edges.copy()

rejected = edges.copy()

keepers = [] # Initalising the boxes that we are to determine

'''

For each contour, we are finding the bounding rectangle and decide if it's one which we require to process

'''

#print contours

#print list(enumerate(contours))

aki = 0

for index\_, contour\_ in enumerate(contours):

if SET1:

print "Processing #%d" % index\_

x, y, w, h = cv2.boundingRect(contour\_)

aki = aki+1

'''

if aki == 1:

img = cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)

cv2.imshow('Akhil',img)

cv2.waitKey(4000)

cv2.destroyAllWindows()

'''

# Checking for the contour and it's bounding box

if keep(contour\_) and include\_box(index\_, hierarchy, contour\_): # If condition satisfied it's referred to satisfy our requirement

keepers.append([contour\_, [x, y, w, h]])

if SET:

cv2.rectangle(processed, (x, y), (x + w, y + h), (100, 100, 100), 1)

#cv2.putText(processed, str(index\_), (x, y - 5), cv2.FONT\_HERSHEY\_PLAIN, 1, (255, 255, 255))

else:

if SET:

cv2.rectangle(rejected, (x, y), (x + w, y + h), (100, 100, 100), 1)

#cv2.putText(rejected, str(index\_), (x, y - 5), cv2.FONT\_HERSHEY\_PLAIN, 1, (255, 255, 255))

new\_image = edges.copy() # Makeing a new white copy of original image

new\_image.fill(255)

'''

cv2.imshow('image',new\_image) # Display the picture

cv2.waitKey(5000) # wait for closing

cv2.destroyAllWindows() # Ok, destroy the window

'''

boxes = []

#print keepers

'''

For each box in our keepers , We need to find the foreground and background intensities of the image.

1.To do so firstly, Find the average intensity of the edge pixels to determine the foreground intensity

2.To Find the intensity of three pixels going around the outside of each corner of the bounding box to determine the background intensity

'''

for index\_, (contour\_, box) in enumerate(keepers):

# 1

fg\_int = 0.0

for p in contour\_:

fg\_int += Intensity(p[0][0], p[0][1]) # sum of intensities of pixels in a counters

fg\_int /= len(contour\_) # Calculating avg. intensity of perticular countour or foreground intensity of our keeper countour

if SET1:

print "FG Intensity for #%d = %d" % (index\_, fg\_int)

#2

x\_, y\_, width, height = box

bg\_int = \

[

Intensity(x\_ - 1, y\_ - 1), #bottom left corner 3 pixels

Intensity(x\_ - 1, y\_),

Intensity(x\_, y\_ - 1),

Intensity(x\_ + width + 1, y\_ - 1), #bottom right corner 3 pixels

Intensity(x\_ + width, y\_ - 1),

Intensity(x\_ + width + 1, y\_),

Intensity(x\_ - 1, y\_ + height + 1), #top left corner 3 pixels

Intensity(x\_ - 1, y\_ + height),

Intensity(x\_, y\_ + height + 1),

Intensity(x\_ + width + 1, y\_ + height + 1), #top right corner 3 pixels

Intensity(x\_ + width, y\_ + height + 1),

Intensity(x\_ + width + 1, y\_ + height)

]

#print np

bg\_int = np.median(bg\_int) #Finding the median of the background intensities

if SET:

print "\n\tBack ground median intensity of the image is given by: ",bg\_int

if SET1:

print "\n\tBG Intensity for #%d = %s" % (index\_, repr(bg\_int))

if fg\_int >= bg\_int: # Determineing whether the box should be inverted or not

fg = 255

bg = 0

else: # Inverting of box

fg = 0

bg = 255

'''

So we are creating final Image (Black and white image here)

Loop through every pixel in the box and color the pixel ,accordingly

'''

for x in range(x\_, x\_ + width):

for y in range(y\_, y\_ + height):

if y >= img\_y or x >= img\_x:

if SET1:

print "pixel out of bounds (%d,%d)" % (y, x)

continue

if Intensity(x, y) > fg\_int:

new\_image[y][x] = bg

else:

new\_image[y][x] = fg

#This is used for image smoothning(blur a bit) so as to improve 'ocr' accuracy

new\_image = cv2.blur(new\_image, (2, 2))

cv2.imwrite(output\_file, new\_image)

if SET:

cv2.imwrite('edges.jpg', edges)

cv2.imwrite('processed.jpg', processed)

cv2.imwrite('rejected.jpg', rejected)

**SCREEN SHOTS**

**NOW WE NEED TO APPLY ENTROPY and PSNR Test to goodness of the Algorithum**

**Sample 1:**

**Image:**

****

**Output:**

****

**OCR Input:**

**on campus? Easi’a’emelye**

**Wa nt to?**

**Friday April 2nd**

**SAU 1 500 Alumni Room**

**12:00 PM**

**W: & Prizes**

**Learn to keep yourself safe**

**& talk about crime prevention with**

**Public Safety & Presenter Tony Yasback.**

**OCR output:**

**Do you feel safe on campus? mum”?**

**Want to?**

**5 l**

**i.- 3’ I Friday April 2nd**

**3 av— SAU 1 500 Alumni Room**

**n**

**w...**

**12:00 PM**

**4' -**

**\*' 1 Izes**

**Learn to keep yourself safe**

**.: talk about crime prevention with**

**PUbIIC Safety .'~ PresenterTony Yasback.**

**Entropy:**

**I1 = imread('advt1.jpg');**

**I2 = imread('advt1out.jpg');**

**J1 = entropy(I1);**

**disp(J1);**

**6.0527**

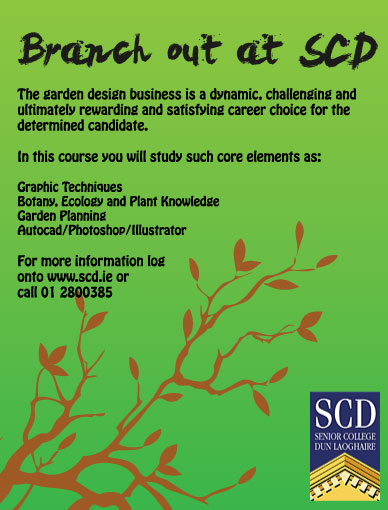
**J2 = entropy(I2);**

**disp(J2);**

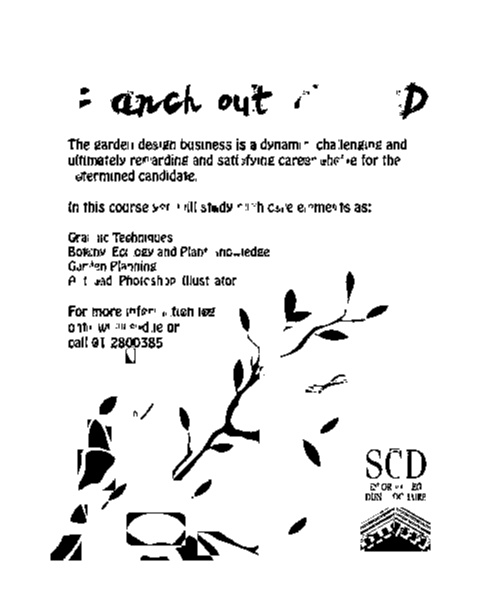
**1.5860**

**Sample 2:**

**Image:**

****

**Output:**

****

**Entropy:**

**I1 = imread('advt2.jpg');**

**I2 = imread('advt2out.jpg');**

**J1 = entropy(I1);**

**disp(J1);**

**6.9424**

**J2 = entropy(I2);**

**disp(J2);**

**1.3078**

**Sample 3**

**Image:**

****

**Output:**

**Sample 4:-**

**Image:**

****

**Entropy:**

**I1 = imread('advt3.png');**

**I2 = imread('advt3out.png');**

**J1 = entropy(I1);**

**disp(J1);**

**4.5512**

**J2 = entropy(I2);**

**disp(J2);**

**0.6995**

**Image:**

****

**Output:**

****

**Entropy:**

**I1 = imread('advt4.jpg');**

**I2 = imread('advt4out.jpg');**

**J1 = entropy(I1);**

**disp(J1);**

**3.2864**

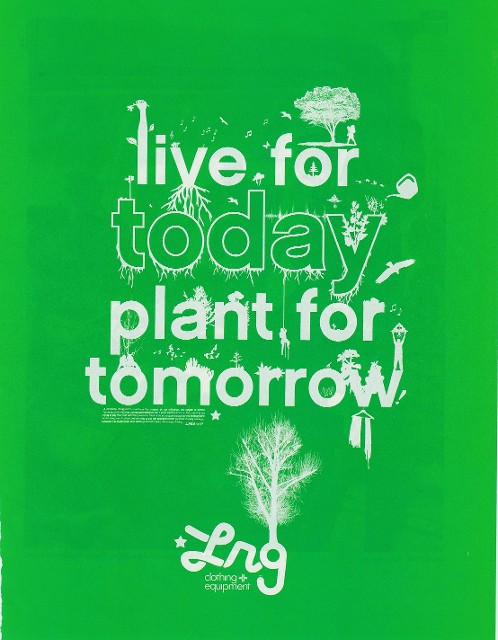
**J2 = entropy(I2);**

**disp(J2);**

**1.0986**

**Sample 5:**

**Image:**

****

**Output:**

****

**Entropy:**

**I1 = imread('advt5.jpg');**

**I2 = imread('advt5out.jpg');**

**J1 = entropy(I1);**

**disp(J1);**

**6.6919**

**J2 = entropy(I2);**

**disp(J2);**

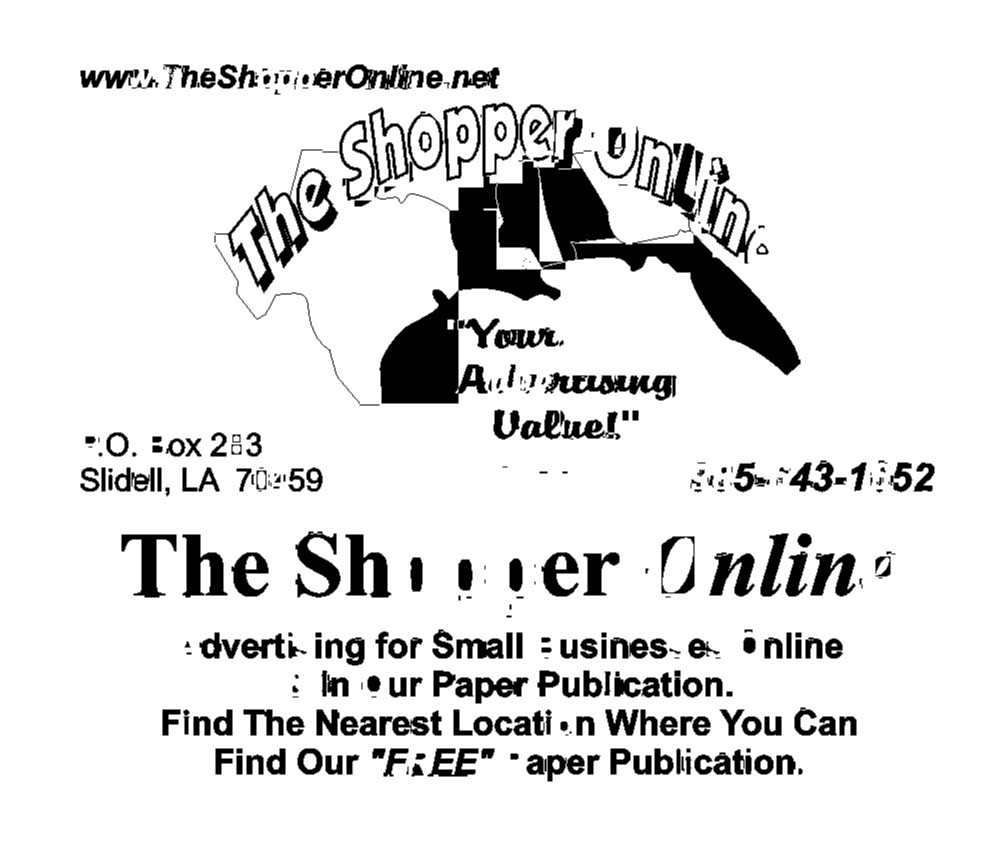
**0.9149**

**Sample 6:**

**Image:**

****

**Output:**

****

**Entropy:**

**I1 = imread('advt7.jpg');**

**I2 = imread('advt7out.jpg');**

**J1 = entropy(I1);**

**disp(J1);**

**3.9495**

**J2 = entropy(I2);**

**disp(J2);**

**1.3388**

**Sample 7:**

**Image:**

****

**Output:**

****

**OCR INPUT:-**

**G45**

**EXPRESS CHIPSET**

**OCR OUTPT:-**

**Lirltgl**

**G45**

**EXPRESS (NIPSET**

**Entropy:**

**I1 = imread('intel2.jpg');**

**I2 = imread('intel2out.jpg');**

**J1 = entropy(I1);**

**disp(J1);**

**2.6879**

**J2 = entropy(I2);**

**disp(J2);**

**2.9709**

**Sample 8:**

**Image:**

****

**Output:**

****

**OCR INPUT:-**

**“a**

**Light Bulb**

**Photography**

**OCR OUTPUT:-**

**®@**

**Light Bulb**

**Photography**

**Swnchlng vou on to Photography**

**Entropy:**

**I1 = imread('new1.jpg');**

**I2 = imread('new1out.jpg');**

**J1 = entropy(I1);**

**disp(J1);**

**1.0492**

**J2 = entropy(I2);**

**disp(J2);**

**1.0589**

**Sample 10:**

**Image:**

****

**Output:**

****

**OCR INPUT:-**

**Q**

**Lorem Ipsum**

**dolor sit omet**

**OCR OUTPUT:-**

**Q**

**‘ WIT**

**{1 um“**

**Lorem psum**

**odor s“ 0 net**

**Entropy:**

**I1 = imread('new.jpg');**

**I2 = imread('newout.jpg');**

**J1 = entropy(I1);**

**disp(J1);**

**1.2955**

**J2 = entropy(I2);**

**disp(J2);**

**1.0507**

**CONCLUSION**

**We have taken Ten Samples and applied Entropy test to both Input Image and Output Image. On getting entropy values of Input Image and Outpput Image, We see that for almost 7 images Entropy of Input image is greater than Entropy of Output Image.For 2 images we get grater image Entropy than input and for image nearly values are reached.**

**So we can conclude that this method is good enough for large text images and not good for small text images.**

**Upon improving box technique we can reduce no.of good rejected boxes thus improving quality and we can extent this technique to Midium text images.**

**We wan’t to extend this technique to improve OCR for any given image.**

**References**

* **J. Canny. A computational approach to edge detection. IEEE trans. PAMI, 8(6):679–698, 1986.**
* **J. N. Kapur, P. K. Sahoo, and A. Wong. A new method for gray-level picture thresholding using the entropy of the histogram. Computer Vision Graphics Image Process., 29:273–285, 1985.**
* **J. Sauvola and M. Pietikainen. Adaptive document image binarization. Pattern Recognition, 33:225–236, 2000.**
* **O. D. Trier and A. Jain. Goal-directed evaluation of binarization methods. IEEE Trans. PAMI, 17(12):1191–1201, 1995.**
* **C. Wolf, J. Jolion, and F. Chassaing. Text localization, enhancement and binarization in multimedia documents.ICPR, 4:1037–1040, 2002.**
* **S. Yanowitz and A. Bruckstein. A new method for image segmentation. Computer Vision, Graphics and Image Processing, 46(1):82–95, 1989.**

**Text book and Resources**

* **W. Niblack. An introduction to digital image processing.**
* **Prentice Hall, pages 115–116, 1986.**
* **OpenCV Computer Vision with Python. Joseph Howse**
* **Learning OpenCV, Gary Bradski and Adrian Kaehler**
* **Programming Computer Vision with Python, Jan Erik Solem ,March 18, 2012**
* **Youtube Lectures on Image Processing and OpenCV**

**Acknowledgement**

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. G.Malathi,** Associate Professor, School of Computer Science, for her consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr. Jegannathan.,** Dean of the School of Computer Engineering, VIT University, for extending the facilities of the School towards our project and for her unstinting support.

We express thanks to **Dr. Ganesan.R,** Programme Chair of School of Computer Engineering, forhis support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

**MARLA AKHIL REDDY** **SAILESH PATRA** **CHAVA SAI TEJA**