# Optical Character Recognition with Feature Determined Region of Interest

Hugo Silva
Columbia University
UNI: ha2385

Divya Kamat
Columbia University
UNI: dpk2123

Hafeni Heita
Columbia University
UNI: hnh2110

***Abstract*—This paper provides a novel system that performs Optical Character Recognition over images specified by the user using real-time hand detection. The OCR functions have been provided by Tesseract, an open source OCR.**

## I. Introduction

The process of character recognition has been a part of modern society for almost a century, predating the modern PC. Digit recognition devices were commercially available by the beginning of the 1930s, and by the 1960s, a sorting system had been implemented by the United States Postal Services that relied on a character recognition system. By the 1970s, banking systems in many major cities across the world relied on character recognition systems to operate more efficiently. Even before the invention of the PC, character recognition was an incredibly useful and lucrative field and continues to be to this day [1].

Since the invention of the PC and the development of smart phones, it was possible for individuals to make use of character recognition software. However, it is one very important factor that makes optical character recognition useful to todays consumer the incredible abundance of cheap, readily available digital cameras. This revelation has opened up the field of optical character recognition for applications that may not be very useful to big businesses. The shift of focus from corporations to individual consumers has created a field of numerous possibilities, and it is for these reasons that we have chosen to undertake a project in this interesting area of research and development.

## II. Project Breakdown

The block diagram of the entire system is given in 1. A detailed breakdown of the project is provided in the following sections.

### A. Development Platform

The development platform we have chosen for the project is C++ with OpenCV. In addition, we will be making use of the Visual Studio integrated development environment (IDE).

*1) Image Capture:* For the image capture, we use a real-time image capturing feature using a Webcam. For the variable rectangle mode, the algorithm first detects the hand position by positioning a rectangle within the hand in order to detect the hand from the background. An image of the hand is first taken. This image is then converted from RGB to HSV scale.

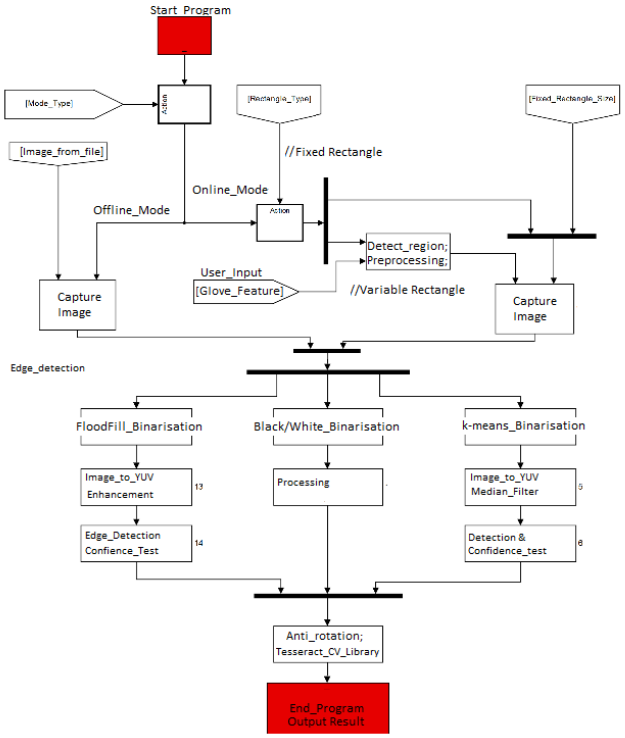



Fig. 1: System Overview

We then ignore the V component and construct a matrix of H and S. Then iterating through the image captured, the values of H and S for each pixel are checked and for the corresponding H and S we insert a 1 into the matrix. This is followed by a morphological closing operation with a 4x4 element. This serves as the filter for recognizing the hand positions real-time. For detecting the actual hand positions within the image, the first step is to perform noise removal using a 31x31 median filter. It then compares the image to the filter to verify if its a part of the hand, and takes the two largest connected components and recognizes those for the hands. Using these hand positions, we then detect the image between the rectangle and crop that to obtain the image for the OCR.

In the fixed rectangle mode, the rectangle for cropping the image is fixed and can be increased or decreased in size to fit the desired image. The image within the given boundaries is then cropped and used for further processing.

### B. Preprocessing

Once an input image has been supplied by the users digital camera, it is necessary to process the image before attempting to match the individual characters the image contains. The input image will be subject to noise from various sources, rotational distortion and error of parallax. Numerous steps need to be taken to correct for these sources of error. Below are the steps with associated implementation algorithms [2].

*1) Contrast Enhancement:* Contrast Enhancement using Brightness Preserving Bi-Histogram Equalization [5]: Histogram equalization is widely used for contrast enhancement in a variety of applications due to its simple function and effectiveness. One drawback of the histogram equalization can be found on the fact that the brightness of an image can be changed after the histogram equalization, which is mainly due to the flattening property of the histogram equalization. Therefore we are using the modified approach of BBHE for performing contrast enhancement. The BBHE firstly decomposes an input image into two sub-images based on the mean of the input image. One of the sub-images is the set of samples less than or equal to the mean whereas the other one is the set of samples greater than the mean. Then the BBHE equalizes the sub-images independently based on their respective histograms with the constraint that the samples in the formal set are mapped into the range from the minimum gray level to the input mean and the samples in the latter set are mapped into the range from the mean to the maximum gray level. This approach preserves the mean brightness of the image and hence doesnt deteriorate the image quality.

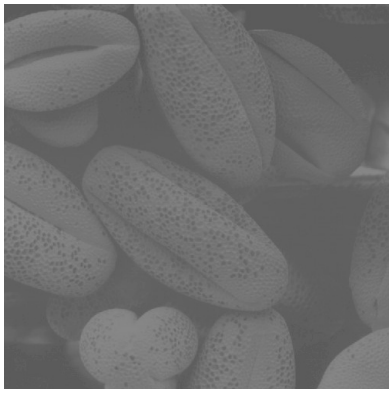


Fig. 2: Original image

*2) Anti Skewing:* Anti skewing is the process of rotational correction. Input images are likely to have some form of rotation due to the method of capture. The first step to correcting this is to discern a measure for the rotation of the input image. Figure 4 shows the example of an input image that has been unintentionally rotated. One method for measuring the level of rotation is to attempt to detect the beginning of the next line every k lines down, where $k > 0$ , $k \in \mathbb{N}$. Once a array of position vectors locating the beginning of every k lines has been collected, a line of best fit can be placed and the gradient of the rotation can be calculated. 4

Fig. 3: Equalized image

shows a rotated image with a line of best fit and a few lengths measured from the edge of the view of the camera to the start of every k lines.

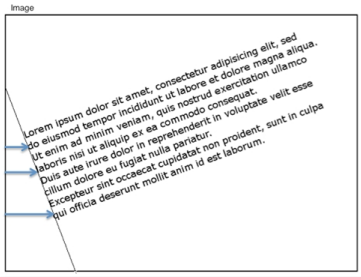



Fig. 4: Rotated image

Once the line of best fit has been placed, the gradient of that line can be calculated. A simple matrix transformation can then be applied to rotate the input image appropriately, correcting the rotational distortion. Algorithm 1 details the implementation of such a method.

**Algorithm 1** Anti Skewing algorithm

**procedure** ANTISKEWING
  $image \leftarrow read\ input\ image$
  $k \leftarrow constant$
  $y \leftarrow []Emptyarray$
  $[y, grad] \leftarrow calculateRotation(image, k)$
  $\theta = arctan(grad)$
  $out \leftarrow rotate(image, -\theta)$
  **function** $[y, grad] = calculateRotation(input, const)$
    **for** $i = 1 : round(\frac{num\_rows}{k})$ **do**
      $y(i) \leftarrow (x_p, y_p)$ ▷ Position of start of image;
    **end for**
    $grad = \frac{y_p(end) - y_p(1)}{x_p(end) - x_p(1)}$
  **end function**
**end procedure**

When implementing the code, one major problem was encountered: the program often attempted to compute the

tangent of 90 degrees which is undefined. This caused the program to effectively crash when the output was correct, which proved problematic. This issue was resolved using a combination of a "last-run" method and the Hough Transform. The Hough transform is a method for computing the gradient of the region of interest within an image. The reason we needed to make the switch to the Hough transform is that it is in polar coordinates. By converting the coordinates to polar coordinates and computing a magnitude, angle pair we were able to represent x, y coordinates by cosines and sines respectively. This allowed us to circumvent this problem of an undefined tangent function.

The "last-runs" method was also a revelation to optimising this section of code. Originally, a gradient was calculated every k lines, however, there was no gaurantee that every k lines would start with a letter pixel which caused some messy, if statements. The last-runs method attempted to detect the final line of letter pixels of each line of text. This was useful as it avoided the problem of contours affecting the local gradient at the boundary of each letter. Reducing each line of text to a single line of pixels allowed the Hough transform to be implemented more effectively.

*3) Parallax Correlation:* Error of parallax occurs when a camera that is not placed directly above the text to be analysed takes an image. As a result, the image is not contained by a rectangle but by either a parallelogram or a trapezoid. The first step towards correcting this image is to first locate the four corners of the shape containing the image and to perform some calculations to identify the shape these four corners compose. In order to reconstruct the image as a rectangle, it is then necessary to shift the (x,y) coordinates of all the pixels as well as interpolate some pixel values to fill in any missing pixels needed when recreating a rectangular image. Figure 5 shows an example of an image that suffers from error of parallax [3].

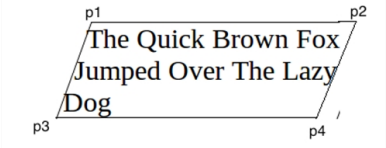



Fig. 5: Image that suffers from errors of parallax

In order to carry out the transformation, algorithm 2 is utilised.

*4) Segmentation:* Since this program is only dealing with computer printed input data, the segmentation algorithm is relatively straightforward. In the first pass of the segmentation algorithm, we will not deal with touching letters (cursive text) however; we will attempt to add this feature into the application. Below is an overview of the segmentation method.

The algorithm will attempt to segment the input image into two main categories, background and text with an appropriately chosen threshold T. Background pixels will be set to white (1) and text to black (0). Once a segmented input has been constructed, the image will undergo closing by an appropriate structuring element, in order to fill holes that may have been created due to noise sources. Holes are defined as random, isolated white or black pixels. Once filling has been carried out. Note: for credit cards, a nearly identical process is carried out, but the segments are instead edge-defined. Through testing, we will discern the appropriate edge-detection method to use on the credit cards and the appropriate thresholding value through testing [3].

We investigate three methods of segmentation: K-means, flood fill and black and white. The objective of these methods is to produce a binary representation of the input image to allow the Tesseract recognition engine to process the text. Below is a brief description of each method.

*a) K-Means:* The K-means method is an iterative process that converges upon an unsupervised learning law. In this program, we have utilised two training clusters, for simplicity let us name them "a" and "b". Each cluster has "d" elements. The program steps through the input image pixel by pixel and computes the euclidean norm between the current pixel

**Algorithm 2** Parallax correlation algorithm

**procedure** PARALLAX
  $P_1 \leftarrow (x_{p1}, y_{p1})$ ▷ Left top corner point;
  $P_2 \leftarrow (x_{p2}, y_{p2})$ ▷ Right top corner point;
  $P_3 \leftarrow (x_{p3}, y_{p3})$ ▷ Left bottom corner point;
  $P_4 \leftarrow (x_{p4}, y_{p4})$ ▷ Right bottom corner point;
  $[m, n] \leftarrow size(image)$
  $out \leftarrow zeros(m, n)$
  $N \leftarrow \frac{((P_3-P_1)+(P_4-P_2))}{2}$
  $M \leftarrow \frac{((P_2-P_1)+(P_4-P_3))}{2}$ ▷ The following are function handles;
  $p_{1j} = \frac{x_{p2}j-(M-j)x_{p1}}{M}$
  $p_{2j} = \frac{x_{p2}i+(N-i)x_{p3}}{N}$
  $p_{3j} = \frac{x_{p3}j+(M-j)x_{p4}}{M}$
  $p_{4j} = \frac{x_{p1}i+(N-i)x_{p4}}{N}$
  $p_{1i} = \frac{y_{p2}j-(M-j)y_{p1}}{M}$
  $p_{2i} = \frac{y_{p2}i+(N-i)y_{p3}}{N}$
  $p_{3i} = \frac{y_{p3}i+(M-j)y_{p4}}{M}$
  $p_{4i} = \frac{y_{p1}i+(N-i)y_{p4}}{N}$ ▷ Utilising the function handles
  **for** $i = 1 : m$ **do**
    **for** $j = 1 : n$ **do** ▷ Calculate $(x, y)$ which will be copied from the input image to position $(i, j)$
      $x = [p_{3j}(p_{3i}p_{1i})(p_{4j}p_{2j}) - p_{3i}(p_{3j}p_{1j})(p_{4j}p_{2j})p_{4j}(p_{4j}p_{2j})(p_{3i}p_{1i}) + p_{4i}(p_{3j}p_{1j})(p_{4j}p_{2j})]\backslash[(p_{3i}p_{1i})(p_{4j}p_{2j}) - (p_{4i}p_{2i})(p_{3j}p_{1j})]$;
      $y = [p_{3i}(p_{3j}p_{1j})(p_{4i}p_{2i}) - p_{3j}(p_{3i}p_{1i})(p_{4i}p_{2i})p_{4i}(p_{4i}p_{2i})(p_{3j}p_{1j}) + p_{4j}(p_{3i}p_{1i})(p_{4i}p_{2i})]\backslash[(p_{3j}p_{1j})(p_{4i}p_{2i}) - (p_{4j}p_{2j})(p_{3i}p_{1i})]$;
      $out(i, j) = image(x, y)$
    **end for**
  **end for**
**end procedure**

and each element of each cluster. Then, the program finds the norm of the smallest magnitude and this denotes the label of the pixel we are processing. The process is repeated 'M' times. This program uses two elements because we require two labels (background or text) in practice, the user can define as many cluster labels as they deem necessary. Termination occurs when either one of two conditions are met. Either, the program has reached its maximum number of iterations, or the number of pixels whose labels require changing from the previous run to the current run is below some discerned threshold. After termination, the program outputs the pixels closest to cluster 'a' as white and the pixels closest to cluster 'b' as black. The nature of the algorithm, and using no more than two clusters ensures some convergence towards the true binary image $x_t$.

*b) Flood fill:* The flood fill segmentation process has several steps. The first involves converting the input image to the YUV colour space for processing. In the YUV colour space, the image enhancement techniques described in the algorithm above are implemented using the BBHE algorithm. Both the colour space transoformation, and the nature of this equalisation function mitigate the effects of ambient brightness on the resultant image. The enhanced image is then transformed back to an RGB image for edge detection and flood filling. The process of flood filling is conceptually simple. The edges detected define the boundaries between the text and the background. The function proceeds to fill in the empty letters in order to produce the binary image for processing.

*c) Black and White:* The black and white segmentation algorithm carries the least computational cost of the three algorithms utilised in this project. As an added bonus, in applicable cases, it is the most accurate method of the three. Unfortunately, the number of cases for which it is applicable as a method are rather limited and it is incredibly difficult to accurately implement this method on real-time RGB images. The input image is converted to grayscale, passed through a filter and then binarized by an adaptive threshold T. Originally, we attempted to implement this method with a constant threshold T, but when changing test images this became ineffective at accurately discerning background from text. In order to improve the accuracy of this method, the switch was made to a locally adaptive threshold. This produced much better results when feeding the resultant images through the Tesseract engine.

### C. Optical Character Recognition

For performing the Optical Character Recognition, the system uses Tesseract. Tesseract is a free software suite available for performing OCR operations and is considered one of the most accurate open source OCR engines currently available.

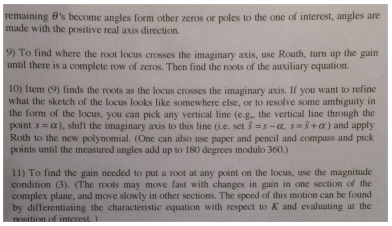
## III. Results

We now present the results of the various algorithms and methods used in this paper as well as a brief description of their reasons for failure.

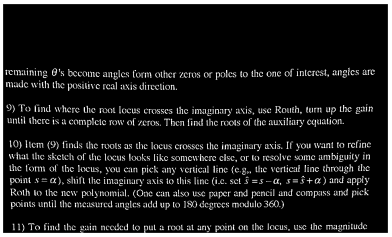
First we examine the Black and White mode of operation. As seen in the figures 6 and 7 the black and white mode works well for documents rather than signs. This happens because most signs have a embossed letters as compared to documents which have simple printed text.

Next we examine the Flood fill mode of operation. As seen in the figures 8 and 9 the flood fill mode does not yield satisfactory results. It works for some cases but fails in most cases.

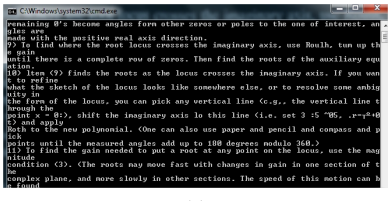
The K-Means mode of operation has the results shown in 10 and 11. As seen in these images, the algorithm works well on clean, noise free images, but for images which have a lot of noise values, the result is garbage values.




(a)




(b)




(c)

Fig. 6: Black and White Mode: (a) Original Image (b) Processed Image (c) OCR Output


(a)


(b)


(c)

Fig. 7: Black and White Failure Case: (a) Original Image (b) Processed Image (c) OCR Output

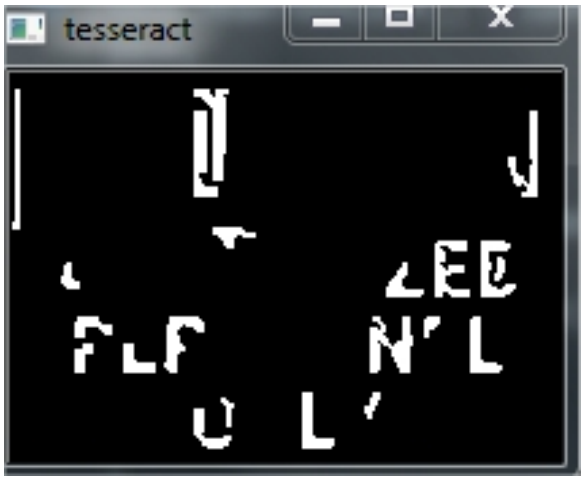

(a)


(b)

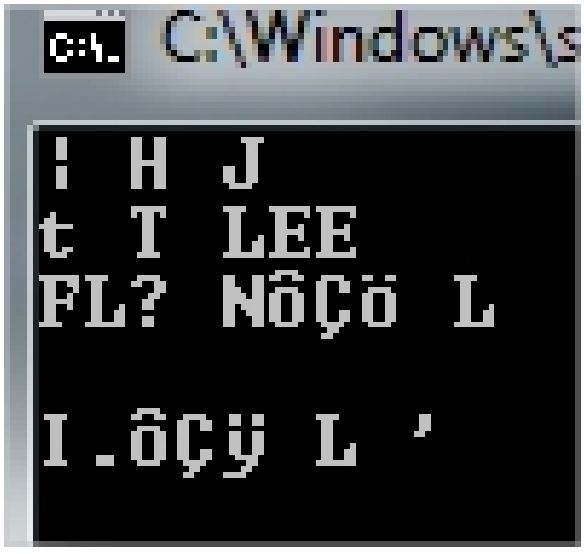

(c)

Fig. 8: Flood Fill Mode: (a) Original Image (b) Processed Image (c) OCR Output


(a)




(b)




(c)

Fig. 9: Flood Fill Failure Case: (a) Original Image (b) Processed Image (c) OCR Output


(a)


(b)


(c)

Fig. 10: K-Means Mode: (a) Original Image (b) Processed Image (c) OCR Output


(a)


(b)


(c)

Fig. 11: K-Means Failure Case: (a) Original Image (b) Processed Image (c) OCR Output

## IV. Conclusion

This project required numerous algorithms to implement. Since the character recognition itself was carried out by the Tesseract engine, the analysis below has been performed on the algorithms has been outlined below.

BBHE was chosen as the enhancement algorithm for because it performs histogram equalisation centred around the mean of the intensity values of the input image. This means that although the shape of the histogram is transformed, the mean intensity value of the histogram is preserved. During testing, BBHE greatly outperformed the standard histogram equalisation method provided in class.

The anti rotation algorithm successfully corrected a few of the test images, however, in some cases problems were encountered. Unfortunately, proper implementation of the algorithm required a more fine-tuned parameter, lambda, which would require either a machine-learning algorithm or more trial and error. This lambda value controls the condition that determines whether the next black pixel belongs to the current line of text or the next line of text. Accidental classification or labeling of pixels then causes this method to rotate the image by the wrong angle, often making the output image worse than the input image. For this reason, this algorithm was not placed into the final coding project.

Segmentation was possibly one of the most challenging aspects of this project. When working with known test images, accurate segmentation was easy. The necessary parameters could be properly tuned, and we saw success with all three methods (black and white, k-means and flood fill). It is difficult to say which method outperforms which in this case. When the environment conditions were correct, the black and white method definitely produced the most recognizable outputs for the Tesseract engine, so in that sense it could be considered the best in some sense. However, the black and white method did not work well at all when it came to colour images and signs, making it an incredibly limited method. This made the algorithm great offline purposes but not incredibly useful for the realtime applications. Next, the flood fill method produced the widest usable domain. When the threshold was properly chosen, this method could deal with both colour and black and white images well, however, tuning the threshold became the collapse of this methods efficacy. Finally the k-means method was the best all-round method. Although it neither produced the best results, nor worked on as many test images as the flood fill method did, it was the best unsupervised method on the widest range of test images.

## Acknowledgment

We would like to thank Prof. David Gibbon and Prof. Zhu Liu for their support and guidance through the course of this project. We would also like to thank Hang Guan and Yu Gan for their help in making this project a success.

## V. References

[1] HIROMICHI FUJISAWA, YASUAKI NAKANO, and KIYOMICHI KURINO. Segmentation Methods for Character Recognition: From Seg-

mentation to Document Structure Analysis. INSTITUTE OF AUTOMATION CAS., November 2009: 1079 - 1091.
[2] Smith, Ray. Tesseract OCR Engine - "What it is, where it came from, where it is going?". OSCON. Google Inc, 2007.
[3] Wojciech Bieniecki, Szymon Grabowski and Wojciech Rozenberg. Image Preprocessing for Improving OCR Accuracy. Lviv-Polyana: MEMSTECH, 2007, 75-80.
[4] "A Spatial Median Filter for Noise Removal in Digital Images", James Church, Dr. Yixin Chen, and Dr. Stephen Rice.
[5] "Contrast Enhancement Using Brightness Preserving Bi-Histogram Equalization", Yeong-Taeg Kim, Member, IEEE.