Form segmentation with tesseract

Tanvir Hasan

Reg No: 12101005

Anup Kumar Kar

Reg No: 12101033

Sayem Hossain

Reg No: 12101046

submitted in partial fulfilment of the degree of

Master of Science

at the University of Asia Pacific, Dhaka,

Bangladesh.

March 26, 2016

Declaration

We, hereby, declare that the work presented in this thesis is the outcome of the investigation performed by us under the supervision of Md. Shiplu Hawlader, Lecturer, Department of Computer Science and Engineering, University of Asia Pacific. We also declare that no part of this thesis and thereof has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

Md. Shiplu Hawlader
Supervisor

Signature

Tanvir Hasan
Candidate

Anup Kumar Kar
Candidate

Sayem Hossain
Candidate

Approval

The Thesis Report " Form segmentation with tesseract" submitted by Tanvir Hasan, Reg.NO.12101005, Anup Kumar Kar, Reg.NO.12101033, Sayem Hossain, Reg.NO.:12101046 students of Spring-2012, to the Department of Computer Science & Engineering, University of Asia Pacific, has been accepted as satisfactory for the partial fulfilment of the requirements for the degree of Bachelor of Science in Computer Science & Engineering and approved as to its style and contents.

Approved as to the style & contents by	
Dr. Bilkis Jamal Ferdosi Assistant Professor Department of Computer Science & Engineering, University of Asia Pacific	
Dr. Khan Md. Anwarus Salam Assistant Professor Department of Computer Science & Engineering, University of Asia Pacific	
Md. Akhtaruzzaman Adnan Assistant Professor Department of Computer Science & Engineering, University of Asia Pacific	
Md. Habibur Rahman Lecturer Department of Computer Science & Engineering, University of Asia Pacific	

Acknowledgements

First of all, thanks to Almighty Allah for giving us the potency and energy to complete this thesis successfully.

We want to express out gratefulness towards our thesis supervisor Md. Shiplu Hawlader for his valuable advices and important suggestions. His regular and active supervision and erudite directions from the beginning to the end were the driving forces for the successful completion of the research work.

We would like to convey our thankfulness to all of our teachers at the Department of Computer Science and Engineering, University of Asia Pacific. Discussions with many of them have helped us tremendously in improving the quality of our work. We also thank the department for providing us with resources which were necessary for the preparation of the thesis.

And last but not the least, we would like to express thanks to our parents and family members for their tremendous support and inspiration.

Abstract

Image segmentation is an important component in many image analysis and computer vision tasks. Particularly, the problem of efficient interactive foreground object segmentation in still images is of great practical importance in image editing and has been the interest of research for a long time. Classical image segmentation tools use either texture, color or edge (contrast) information for the purpose of segmentation. Deformable models, Graph-cut, GrabCut etc. are some prominent methods used for the segmentation of a foreground object. Object segmentation methods have helped in many computer vision areas, such as scene representation & interpretation, content based image retrieval, object tracking in videos, medical applications etc.

Most object segmentation techniques in computer vision are based on the principle of boundary detection. These segmentation techniques assume a significant and constant gray level change between the object(s) of interest and the background. However, this is not true in the case of textured images. In textured images, there exist many local edges of the texture micro units (texels), due to the basic nature of a texture image. In case of textured images, the object bound- ary is defined as the place where texture property changes. So to perform the correct segmentation in case of textured images, there is a need to incorporate the textural information in the segmentation process.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Segmentation of Images [1] is the widely investigated eld of image processing, image analysis and important module of early vision problem. It is the process to cluster a form image into some isolated image regions corresponding to individual surface, objects or some natural part of the object. It is the process of separating an image into some disjoint or distinct regions whose characteristic such as intensity; colon texture etc. are similar. No two such regions are similar with respect to these characteristics. in digital image processing, digital image analysis usually involves a 'low-level' and a 'high-level' processing. In low-level analysis, the representation of an image is transformed from a numerical array of pixel intensities to a symbolic set of image primitives: edges and regions. In high-level analysis, object labels (or interpretations) are assigned to these primitives, thereby providing a semantic description of the image.

An image is segmented [1] for different kind of implementations like object recognition to extract data from it, occlusion boundary estimation within motion or stereo systems, image compression, image editing or image database lookup.

The output of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image. Each of the pixels in a region are similar with respect to some characteristic or computed pAn image is segmented for different kind of implementations like object recognition to extract data from it, occlusion boundary estimation within motion or stereo systems, image compression, image editing or image database lookup.

property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristic. image segmentation is a fundamental part of the 'low level' aspects of computer visions and has many practical applications such as in medical imaging, industrial automation and satellite imagery.Traditional

methods for image segmentation have approached the problem either from localization in class space using region information or from localization in position, using edge or boundary information. for monochrome images generally are based on one of two basic properties of gray- level values: discontinuity and similarity. In the rst category, the approach is to partition an image based on abrupt changes in gray level. The principal areas of interest within this category are detection of isolated points and detection of lines and edges in an image. The principal approaches in the rst category are based on edge detection, and boundary detection. Basically, the idea underlying most edge-detection techniques is the computation of a local derivative operator. The rst derivative of the gray-level prole is positive at the leading edge of a transition, negative at the trailing edge, and zero in areas of constant gray level. Hence the magnitude of the rst derivative can be used to detect the presence of an edge in an image. In this paper we are proposing a form segmentation, an image segmentation methodology using threshold techniques that will be applied to a form image and data matching algorithms to detect different objects and clusters. The goal of segmenting a form image is to extract text data isolating texts of different part of that image by segmenting form into several sections. It becomes easy to extract digital data by processing individual objects into texts if we can first isolate that object scenario and use it for processing.

1.1 Objective

The objective is to decompose the image into parts for further analysis. In simple cases, the environment might be well enough controlled so that the segmentation process reliably extracts only the parts that need to be analyzed further. The segmentation is reliable provided that the person's clothing or room background does not have the same color components as a human face. In complex cases, such as extracting a complete road network from a grayscale aerial image, the segmentation problem can be very difficult and might require application of a great deal of domaina building knowledge. Another objective of segmentation is to perform a change of representation. The pixels of the image must be organized into higher-level units that are either more meaningful or more efficient for further analysis (or both). A critical issue is whether or not segmentation can be performed for many different domains using general bottom-up methods that do not use any special domain knowledge. This chapter presents segmentation methods that have potential use in many different domains. Both region-based and curve-

based units are discussed in the following sections. The prospects of having a single segmentation system work well for all problems appear to be dim. Experience has shown that an implementor of machine vision applications must be able to choose from a toolset of methods and perhaps tailor a solution using knowledge of the application.

1.2 Motivation

In recent years, with the advancement of digital era, we are facing a problem of converting handwritten data into digital data for the purpose of saving them in database or repositories in order to analyse or perform operation on the data or keeping log for the future. We still need to process handwritten documents and forms manually that takes lot of times and resources and increases the possibilities to make mistake when processing that data. That causes serious Inefficiency and pain worth not to do it manually.

Several offices including government and non-government organizations gets a lot of applications and different kind of forms every day that needs to be processed immediately with efficient measure and more accuracy. In this paper we have proposed an efficient way to process those analogue data into digital texts through form segmentation, a technique of processing form image into digital text. Experimental results shows it?s accuracy and efficiency to process a form that is lot less time consuming and effective.

1.3 Overview of this book

In this chapter we have introduced the problem of predicting amino acid interaction network in protein. Rest of the chapters are organized as follows.

In Chapter 2, we will acquire some background knowledge by discussing about protein structure including amino acid, primary, secondary and tertiary structure of protein.

In Chapter 3, we are going to discuss about some existing researches on protein structure prediction and amino acid interaction network prediction.

In Chapter **??**, we will discuss about amino acid interaction network and verify network properties of amino acid in protein with PDB data.

In Chapter **??**, we are going to formulate the amino acid interaction network prediction problem theoretically and mathematically.

In Chapter **??**, we will present a new algorithm based on multi-objective optimization and ant colony optimization to predict the formulated problem of amino acid interaction network prediction.

In Chapter **??**, we will analyze the algorithm with some PDB data and show the result.

Finally, Chapter **??** concludes the document.

Chapter 2

Background Study

Base of a good research is the understanding of the background terms and definition. To understand the Image processing and character reorganisation, one have to clearly understand about Image filtering, Image Transformations and tesseract OCR Engine operation. In this chapter as background knowledge discovery we will discuss about some Image processing term and tesseract.

2.1 Gray Scale

In photography and computing, a grayscale or greyscale digital image is an image in which the value of each pixel is a single sample, that is, it carries only intensity information. Images of this sort, also known as black-and-white, are composed exclusively of shades of gray, varying from black at the weakest intensity to white at the strongest. [2]

2.2 Blur

Blurring is a very powerful operation used in image processing and procedural texture generation. Blurs involve calculating weighted averages of areas of pixels in a source image for each pixel of the final blurred image. Blurring images so they become fuzzy may not seem like a useful operation, but actually is very useful for generating background effects and shadows.[3]

2.2.1 Gaussian smoothing

The Gaussian smoothing operator is a 2-D convolution operator that is used to 'blur' images and remove detail and noise. The idea of Gaussian smoothing is to use this 2-D distribution as a 'point-spread' function, and this is achieved by convolution. Since the image is stored as a collection of discrete pixels we need to produce a discrete approximation to the Gaussian function before we can perform the convolution. I Gaussian smoothing technique is widely used effect in graphics software, typically to reduce image noise and reduce detail. [3]

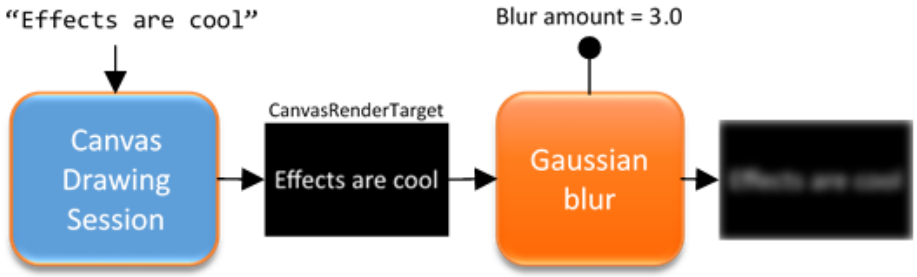


Figure 2.1: Gaussian smoothing

2.3 Thresholding

Thresholding is one of the most basic techniques for what is called Image Segmentation. When the threshold technique is applied on an image, we get segments inside the image and each of segments are represent something.[4]

- Thresholding is a simple way of segmentation.
- We separate out various regions of an image regarding to objects which we want to analyze. The object pixels and the background pixels are the backbone of this separation.
- To differentiate the pixels we are interested in from the rest, we perform a comparison of each pixel intensity value with respect to a threshold.
- Once we have separated properly the important pixels, we can set them with a determined value to identify.

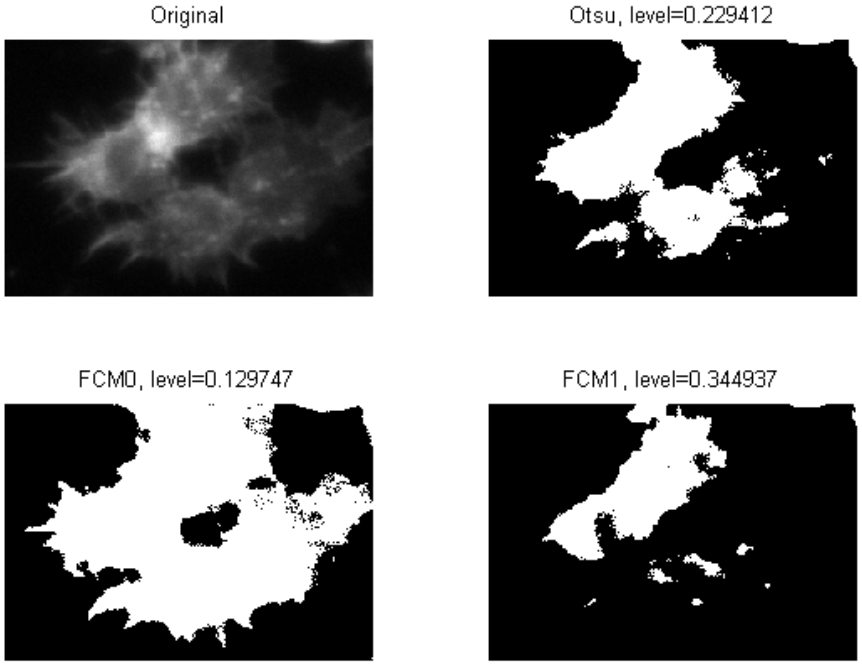
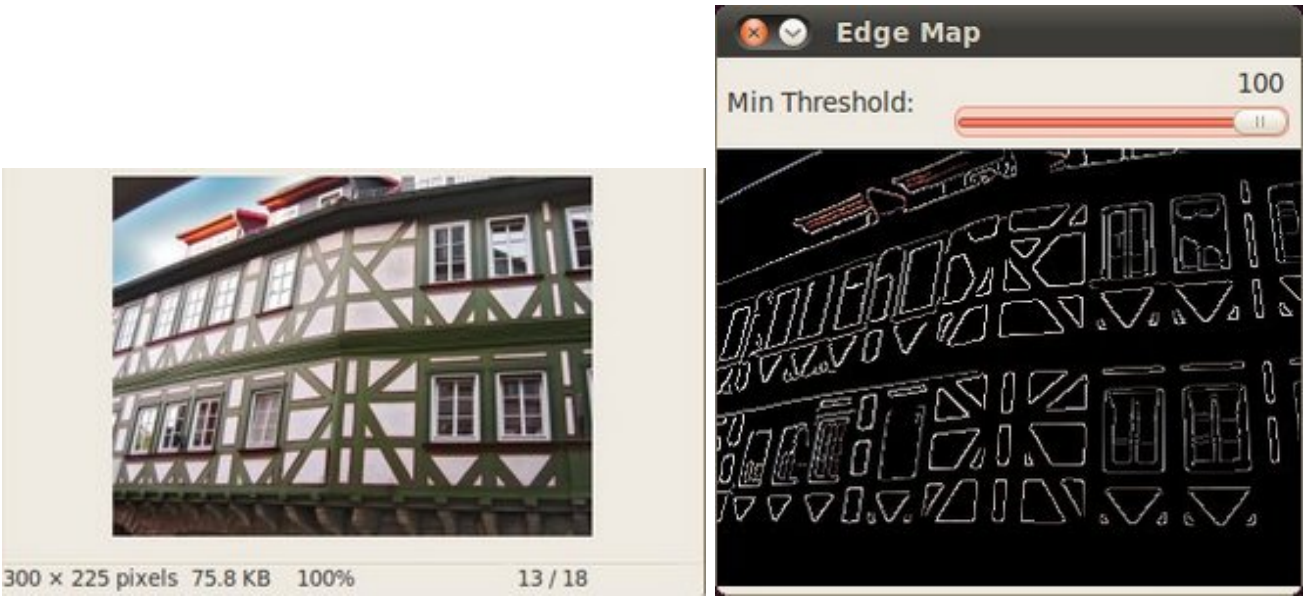


Figure 2.2: Thresholding

2.4 Edge Detection

In digital image processing, edge detection is an important subject matter. Edge detection is a crucial step in object recognition. It is a process of finding sharp discontinuities in an image. The discontinuities are abrupt changes in pixel intensity which characterize boundaries of objects in a scene. In short, the goal of edge detection is to produce a line drawing of the input image. [5]

The Canny operator is also known as the optimal detector, developed by John F. Canny in 1986. There are multiple steps to implement the Canny operator. First, a Gaussian filters is used to smooth the image to remove noise in an image. Second, compute the gradient magnitude. Third, apply the algorithm to remove the pixels that are not part of an edge. Last step is involved with the use of hysteresis thresholding along edges. Hysteresis is based on two thresholds which are upper and lower. If a pixel gradient is higher than the upper threshold, then the pixel will be marked as an edge and if a pixel gradient is below the lower threshold, then the pixel will be marked as a non-edge. [6]

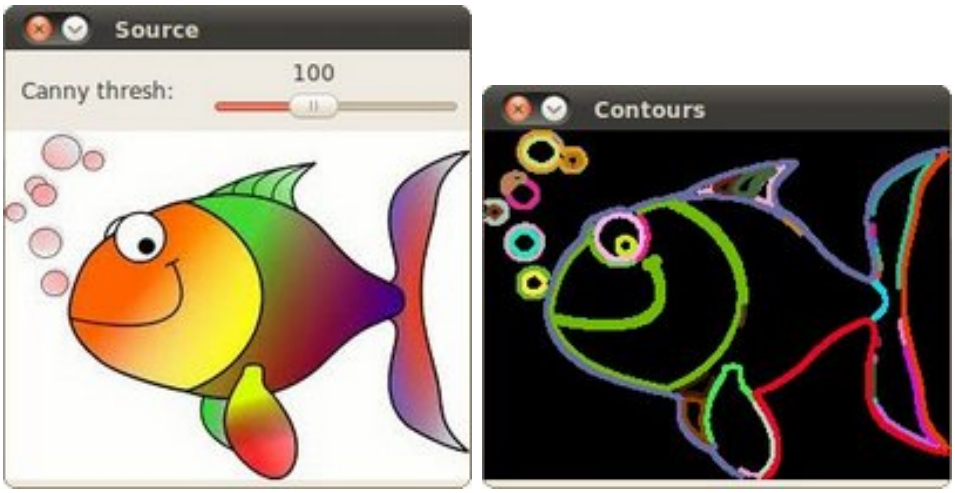


(a) Original Image (b) Edge Detected Image

Figure 2.3: Edge Detection

2.5 Contour

Although algorithms like the Canny edge detector can be used to find the edge pixels that separate different segments. The next step is to be able to assemble those edge pixels into contours. A contour is a list of points that represent, in one way or another, a curve in an image. This representation can be different depending on the circumstance at hand. There are many ways to represent a curve. Contours are represented in OpenCV by sequences in which every entry in the sequence encodes information about the location of the next point on the curve.



(a) Original Image (b) Contour Detect

Figure 2.4: Find Contour

2.6 Tesseract OCR engine

Tesseract is an open source optical character recognition (OCR) engine. HP originally was originally started it as a project. Later it was modified, improved and taken over by Google and later released as open source in year 2005. [7] [8]

Tesseract is considered as one of the most accurate free software OCR engines currently available. A large variety of other OCR software now uses it as a base. It is an excellent quality OCR program, with a large amount of flexibility, a solid codebase, and a large, engaged community of interested people around it.

This thesis investigates the principles of optical character recognition used in the Tesseract OCR engine and techniques to improve its efficiency and runtime.

Optical character recognition (OCR) method has been used in converting printed text into editable text in various applications over a variety of devices such as Scanners, computers, tablets etc. But now Mobile is taking over the computer in all the domains but OCR still remains one not so conquered field. This paper focuses on improving the Tesseract OCR efficiency for Bangla.

This thesis presents a preprocessing technique being applied to the Tesseract Engine to improve the recognition of the characters keeping the runtime low.

2.7 Why Tesseract

The first relevant criterion in Tesseract is the fact that is free and open source (FOSS), which is an advantage and a key point in the research development.

Usually, whenever Tesseract is compared to another free OCR tool, it is the best whether in terms of recognition rate or speed. Even, when it is compared with the Finereader commercial tool, Tesseract arrives to rub it and managed to overtake for handwritten writing.

2.8 Optical character recognition systems

An OCR system is a system that takes a text image as input and applies certain treatments through modules making up the system in order to output editable file with the same text.

The architecture of an OCR system varies from one system to another as needed.Optical Character Recognition systems are usually composed of the following phases:

- Preprocessing phase: prepares the sensor data to the next phase. It is a set of treatments allowing image quality increasing.
- Segmentation phase: delimits document elements (line, word, character). By applying good segmentation techniques, we can increase the performance of OCR systems.
- Feature extraction phase: defines features characterizing the delimited elements of a document. Feature extraction is one of the most important steps in developing a classification system. This step describes the various features characterizing the delimited elements of a document.
- Classification phase: recognizes and identifies each element. It is performed based on the extracted features.
- Post-processing phase: it is an optional phase. It may be automatic or manual.

Several approaches and techniques have been developed for each module.

Chapter 3

Literature Review

The From segmentation process is related to Edge Ditection, Finding contours and contours shapes.And Text ditection related to Tesseract and Tesseact training. Thus in this chapter we will discuss about some state of Image processing technique and Tesseract OCR applications.

3.1 Canny Edge Ditection

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny also produced a computational theory of edge detection explaining why the technique works. [5] [9]

3.1.1 Canny Edge Ditection Algorithm

Step 1 In order to implement the canny edge detector algorithm, a series of steps must be followed. The first step is to filter out any noise in the original image before trying to locate and detect any edges. And because the Gaussian filter can be computed using a simple mask, it is used exclusively in the Canny algorithm. Once a suitable mask has been calculated, the Gaussian smoothing can be performed using standard convolution methods. A convolution mask is usually much smaller than the actual image. As a result, the mask is slid over the image, manipulating a square of pixels at a time. The larger the width of the Gaussian mask, the lower is the detector's sensitivity to noise. The localization error in the detected edges also increases slightly as the Gaussian width is increased. The Gaussian mask used in my implementation is shown below.

$\frac{1}{115}$

2	4	5	4	2
4	9	12	9	4
5	12	15	12	5
4	9	12	9	4
2	4	5	4	2

Figure 3.1: Discrete Approximation to Gaussian function with $\sigma = 1.4$

Step 2 After smoothing the image and eliminating the noise, the next step is to find the edge strength by taking the gradient of the image. The Sobel operator performs a 2-D spatial gradient measurement on an image. Then, the approximate absolute gradient magnitude (edge strength) at each point can be found. The Sobel operator uses a pair of 3x3 convolution masks, one estimating the gradient in the x-direction (columns) and the other estimating the gradient in the y-direction (rows). They are shown below:

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

The magnitude, or EDGE STRENGTH, of the gradient is then approximated using the formula: $|G| = |Gx| + |Gy|$

Step 3 Finding the edge direction is trivial once the gradient in the x and y directions are known. However, you will generate an error whenever sumX is equal to zero. So in the code there has to be a restriction set whenever this takes place. Whenever the gradient in the x direction is equal to zero, the edge direction has to be equal to 90 degrees or 0 degrees, depending on what the value of the gradient in the y-direction is equal to. If GY has a value of zero, the edge direction will equal

0 degrees. Otherwise the edge direction will equal 90 degrees. The formula for finding the edge direction is just:

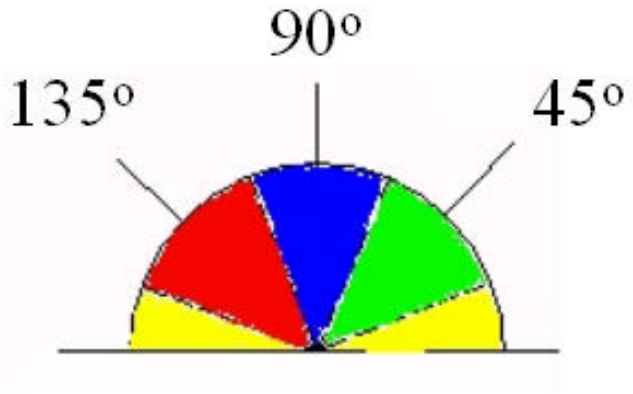
$$\theta = tan^{-1}(Gy/Gx)$$

Step 4 Once the edge direction is known, the next step is to relate the edge direction to a direction that can be traced in an image. So if the pixels of a 5x5 image are aligned as follows:

x	x	x	x	x
x	x	x	x	x
x	x	x	x	x
x	x	a	x	x
x	x	x	x	x
x	x	x	x	x

Table 3.1: Edge Ditection

Then, it can be seen by looking at pixel "a", there are only four possible directions when describing the surrounding pixels - 0 degrees (in the horizontal direction), 45 degrees (along the positive diagonal), 90 degrees (in the vertical direction), or 135 degrees (along the negative diagonal). So now the edge orientation has to be resolved into one of these four directions depending on which direction it is closest to (e.g. if the orientation angle is found to be 3 degrees, make it zero degrees). Think of this as taking a semicircle and dividing it into 5 regions.



Therefore, any edge direction falling within the yellow range (0 to 22.5 & 157.5 to 180 degrees) is set to 0 degrees. Any edge direction falling in the green range

(22.5 to 67.5 degrees) is set to 45 degrees. Any edge direction falling in the blue range (67.5 to 112.5 degrees) is set to 90 degrees. And finally, any edge direction falling within the red range (112.5 to 157.5 degrees) is set to 135 degrees.

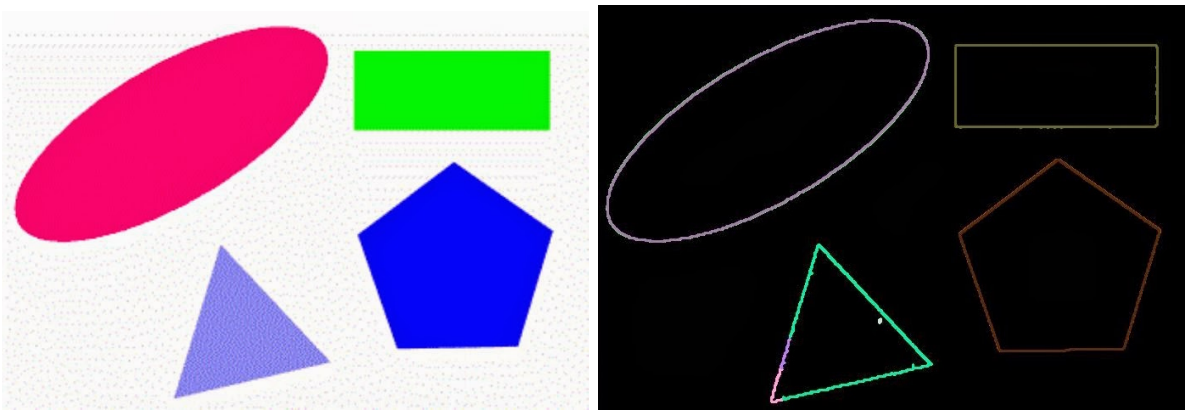
Step 5 After the edge directions are known, nonmaximum suppression now has to be applied. Nonmaximum suppression is used to trace along the edge in the edge direction and suppress any pixel value (sets it equal to 0) that is not considered to be an edge. This will give a thin line in the output image.

Step 6 Finally, hysteresis is used as a means of eliminating streaking. Streaking is the breaking up of an edge contour caused by the operator output fluctuating above and below the threshold. If a single threshold, T1 is applied to an image, and an edge has an average strength equal to T1, then due to noise, there will be instances where the edge dips below the threshold. Equally it will also extend above the threshold making an edge look like a dashed line. To avoid this, hysteresis uses 2 thresholds, a high and a low. Any pixel in the image that has a value greater than T1 is presumed to be an edge pixel, and is marked as such immediately. Then, any pixels that are connected to this edge pixel and that have a value greater than T2 are also selected as edge pixels. If you think of following an edge, you need a gradient of T2 to start but you don't stop till you hit a gradient below T1. [10][11]

3.2 Contours

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition. [12]

- For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
- findContours function modifies the source image. So if you want source image even after finding contours, already store it to some other variables.
- In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.



(a) Original Image (b) Detected Contours

Figure 3.2: Find Courours

3.3 Detecting shapes in images

Doing image processing and especially blob analysis it is often required to check some objects' shape and depending on it perform further processing of a particular object or not. For example, some applications may require finding only rectangle from all the detected objects, or quadrilaterals, rectangles, etc. [13]

3.3.1 Detection quadrilaterals

Detection of quadrilaterals and triangles has pretty much the same idea - we are checking mean distance between provided shape's edge pixels and the edge of estimated quadrilateral/triangle. The only difference here is the way how to estimate parameters of the shape we want to recognize and how to check distance to the estimated shape.

Let's start with quadrilaterals first. For a given shape we can make an assumption that it is a quadrilateral, find its four corners and then using similar method as we've used for circles we can check if our assumption is correct or not - check how good the shape fits into the quadrilateral with assumed parameters.

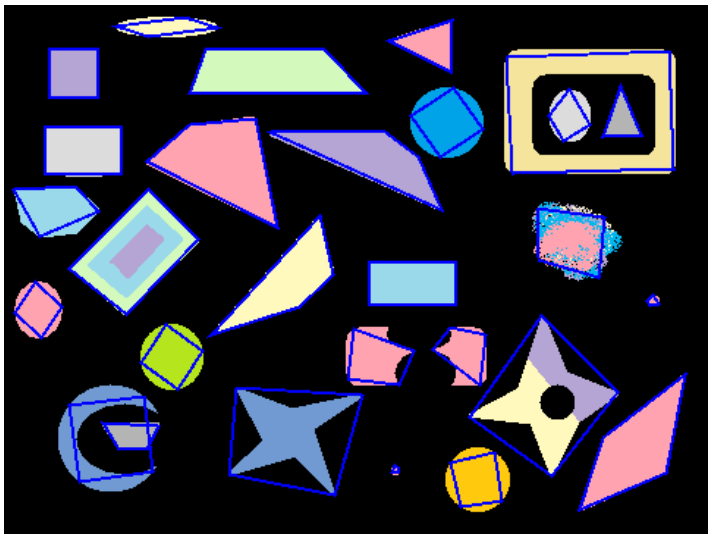


Figure 3.3: Detection of quadrilaterals

As we can see on the image above, we may have different objects and quadrilateral finder provides different results for them. For shapes which really look like quadrilateral, the quadrilateral finder is able to find their corners more or less correctly (problem may occur in the case if object has rounded corners). But for other types of objects (circles, starts, etc.), quadrilateral finder does not provide any good result. And this is correct, since this routine supposes the given shape is really quadrilateral.

Now, when we made the assumption that a subjected object is quadrilateral and we got its corner, we just need to see how good is the fit of the object into the quadrilateral with those found corner - we need to make sure there are no edge points of the given shape which are too far away from the edge of the assumed quadrilateral.

3.3.2 Detection Rectangles, squares, equilateral

Once we made a check for quadrilateral shape and got its 4 corners, we may do some further checks to detect sub-type of the quadrilateral: trapezoid, parallelogram, rectangle, rhombus or square. These checks are based on checking angles between opposite/adjacent sides and their length. First we check two opposite sides - if they are parallel, then we get at least trapezoid. Then we check two other sides - if they are parallel two, then we have parallelogram. If two adjacent sides are perpendicular, then we got rectangle. The last check is to compare length of two adjacent sides - if they are equal, then parallelogram becomes rhombus and rectangle becomes square. And of course we need to add small possible error, so if angle between lines equals to 2 degrees, they are still treated as parallel. [13]

3.4 Tesseract OCR Engine

3.4.1 What is Tesseract?

An Overview of the Tesseract OCR Engine describes Tesseract as: "Tesseract is an open source optical character recognition(OCR) engine. HP originally was originally started it as a project. Later it was modified, improved and taken over by Google and later released as open source in year 2005. It is now available at" (Smith, 2007) . It is very portable as compared to others and supports various platforms. Its focus is more towards providing less rejection and improved accuracy. Currently only command base version is available but there are many projects with UI built on top of it which could be forked. As of now Tesseract version 3.02 is released and available for use. Now Tesseract is developed and maintained by Google. It provides support for around 139 languages.

3.4.2 Architecture

Tesseract OCR is an elegant engine with various layers. It works in step by step manner as shown in the block diagram in fig:3.4 The first step in the cycle is to sense the color intensities of the image, named as adaptive thresholding, and converts the image into binary images. Second step is to do the connected component analysis of the image, which does the task of extracting character outlines. This step is the main process of this cycle as it does the OCR of image with white text and blacks rest of the image. Tesseract was probably the first to use these cycles to process the input image. After this the outlines extracted from image are converted into Blobs(Binary Long Objects). It is then organized as lines and regions and further analysis is for some fixed area. After extraction the extracted components are chopped into words and delimited with spaces. Recognition in text then starts which is a two pass process. As shown in fig 1, the first part is when attempt to recognize each word is made. Each satisfactory word is accepted and second pass is started to gather remaining words. This brings in the role of adaptive classifier. The adaptive classifier then will classify text in more accurate manner. The adaptive classifier needs to be trained beforehand to work accurately. When the classifier receives some data, it has to resolve the issues and assign the proper place of the text.

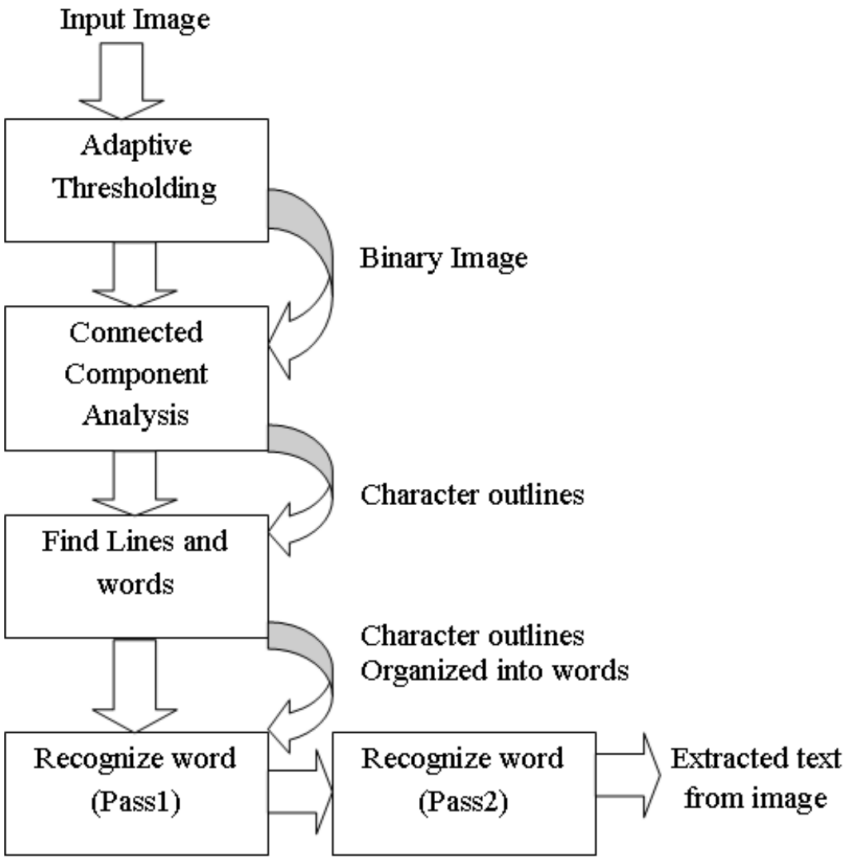


Figure 3.4: Tesseract Flow

3.4.3 Working of Tesseract

Tesseract works pretty much as a scanner. Its interface is pretty simple as it takes input o the command lines with very basic commands. We need to input any image with text in it.

The basic Tesseract command takes only two arguments: First is input image that contains text and second argument is output text file which is usually text file.

Tesseract by default picks the extension of output file as .txt. There no need to specify the explicitly the output file extension.Tesseract supports various languages. Each language comes with a trained language data file.

The language file must be kept in a location Tesseract knows. When using in the project it is advised to keep it within the project folder. This folder is Tesseract home folder in your machine. In this research, we are aiming to extract English and Bangla characters from the images so we have to keep both Bangla and English data files.

3.4.4 How does Tesseract work?

The following is a brief overview of how Tesseract works:

1. Outlines are analysed and stored
2. Outlines are gathered together as Blobs
3. Blobs are organized into text lines
4. Text lines are broken into words
5. First pass of recognition process attempts to recognize each word in turn
6. Satisfactory words passed to adaptive trainer
7. Lessons learned by adaptive trainer employed in a second pass, which attempts recognize the words that were not recognized satisfactorily in the first pass
8. Fuzzy spaces resolved and text checked for small caps
9. Digital texts are outputted

During these processes, Tesseract uses:

1. algorithms for detecting text lines from a skewed page
2. algorithms for detecting proportional and non proportional words (a proportional word is a word where all the letters are the same width)
3. algorithms for chopping joined characters and for associating broken characters
4. linguistic analysis to identify the most likely word formed by a cluster of characters
5. two character classifiers: a static classifier, and an adaptive classifier which employs training data, and which is better at distinguishing between upper and lower case letters

3.4.5 Limitations of Tesseract

Tesseract is an OCR engine, not a complete OCR program. Tesseract is an OCR engine rather than a fully featured program similar to commercial OCR software such as Nuances Omnipage. It was originally intended to serve as a component part of other programs or systems. Although Tesseract works from the command line, to be usable by the average user the engine must be integrated into other programs or interfaces, such as FreeOCR.net, WeOCR or OCRpous. Without integration into programs such as these, Tesseract has no page layout analysis, no output formatting and no graphical user interface (GUI).

3.5 Training Tesseract

The setup for English & Bangla OCR within Tesseract is another important task. The way train the engine and input the dataset matters a lot. Training Tesseract for English & Bangla includes four steps mainly and they are as follows:

3.5.1 Generating training images

Tesseract official website explains these steps very well "The first step is to determine the full character set to be used, and prepare a text or word processor file containing a set of examples" [31]. The two important points to remember for a training file are: Firstly make sure the file contains all the characters that we are expecting. Secondly there should be at least 5 samples. There should be more samples of the more frequent characters - at least 20 [31]. Keeping all this in mind let's explore how do we create box files.

3.5.2 Make box files

Box file is defined as a sample to match with the characters. We need a 'box' file for each training image. Tesseract official website defines the box file as "The box file is a text file that lists the characters in the training image, in order, one per line, with the coordinates of the bounding box around the image" [31]. Tesseract is not so intuitive about the sample data. Inconsistencies may lead to wrong interpretation of data.

3.5.3 Run Tesseract

In this step we run the Tesseract engines with the new training files. The purpose is to create the trained dataset which works as a rule engine. Also it creates log files.

3.5.4 Compute character set

Guidelines from Tesseract website state that "Tesseract needs to know the set of possible characters it can output. To generate the unichar set data file, use the unicharset_extractor program on the box files generated above":extractor language. fontname. box One more requirement in this is, Tesseract needs to have access to character properties i.e. isalpha, isdigit, isupper, islower, ispunctuation [31]. The system described in [30],[31] have classified Hindi OCR and its setup, training very nicely. The only limitation is that the system is for desktop. We have to build the similar system for mobile devices.

3.6 OpenCV

OpenCV[1] (Open Source Computer Vision) is a library of programming functions and algorithms [36]. Their main focus is to provide API mainly aimed at real-time computer vision. It was originally developed by Intel. OpenCV library is free for use for development (under the open source BSD license) [36]. The best feature is that the library is cross-platform .

References

[1] Dr. Jenila Livingston L.M. Santosh. Text detection from documented image using image segmentation. *INTERNATIONAL JOURNAL OF TECHNOLOGY ENHANCEMENTS AND EMERGING ENGINEERING RESEARCH*, 1, 2013.

[2] Wiki. Grayscale. https://en.wikipedia.org/wiki/Grayscale.

[3] OpenCv. Smoothing images. http://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html.

[4] Wiki. Thresholding. https://en.wikipedia.org/wiki/Thresholding_(image_processing).

[5] Robin S M Chrystie. Edge detection in image processing. http://cuens.soc.srcf.net/img/Academic_Material/Edge_Detection.pdf.

[6] OpenCV. Canny edge detection in opencv. http://docs.opencv.org/trunk/da/d22/tutorial_py_canny.html.

[7] Ray Smith. An overview of the tesseract ocr engine. http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/33418.pdf.

[8] Tesseract. Documentation. https://github.com/tesseract-ocr/tesseract/wiki/Documentation.

[9] Wiki. Canny edge detector. https://en.wikipedia.org/wiki/Canny_edge_detector.

[10] Canny edge detector. http://homepages.inf.ed.ac.uk/rbf/HIPR2/canny.htm.

[11] OpenCV. Canny edge detector. http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html.

[12] OpenCV. Contours. http://docs.opencv.org/trunk/d4/d73/tutorial_py_contours_begin.html.

[13] Detecting simple shapes in an image. http://opencv-code.com/tutorials/detecting-simple-shapes-in-an-image/.