

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SCHOOL OF SCIENCE AND TECHNOLOGY**

**COURSEWORK FOR THE BSC (HONS) COMPUTER SCIENCE; YEAR 2**

**COURSEWORK FOR THE BSC (HONS) INFORMATION TECHNOLOGY; YEAR 2**

**ACADEMIC SESSION AUGUST 2019;**

**CSC2014: DIGITAL IMAGING PROCESSING**

**DEADLINE: 18TH NOVEMBER 2019 (MONDAY), 5:00 PM**

**STUDENT NAME:** Lee Pui Ling Elizabeth, Sasnitha Kandasamy, Nimue Wafiya,

Hiba Azhari

**NRIC/PASSPORT NO:** 18082784, 17043977, 17009952, 16065005

**INSTRUCTIONS TO CANDIDATES**

# This assignment will contribute 20% to your final grade.

* This is a group (up to 4 people) assignment.

**IMPORTANT**

# The University requires students to adhere to submission deadlines for any form of assessment. Penalties are applied in relation to unauthorized late submission of work.

# Coursework submitted after the deadline but within 1 week will be accepted for a maximum mark of 40%.

# Work handed in following the extension of 1 week after the original deadline will be regarded as a non-submission and marked zero.

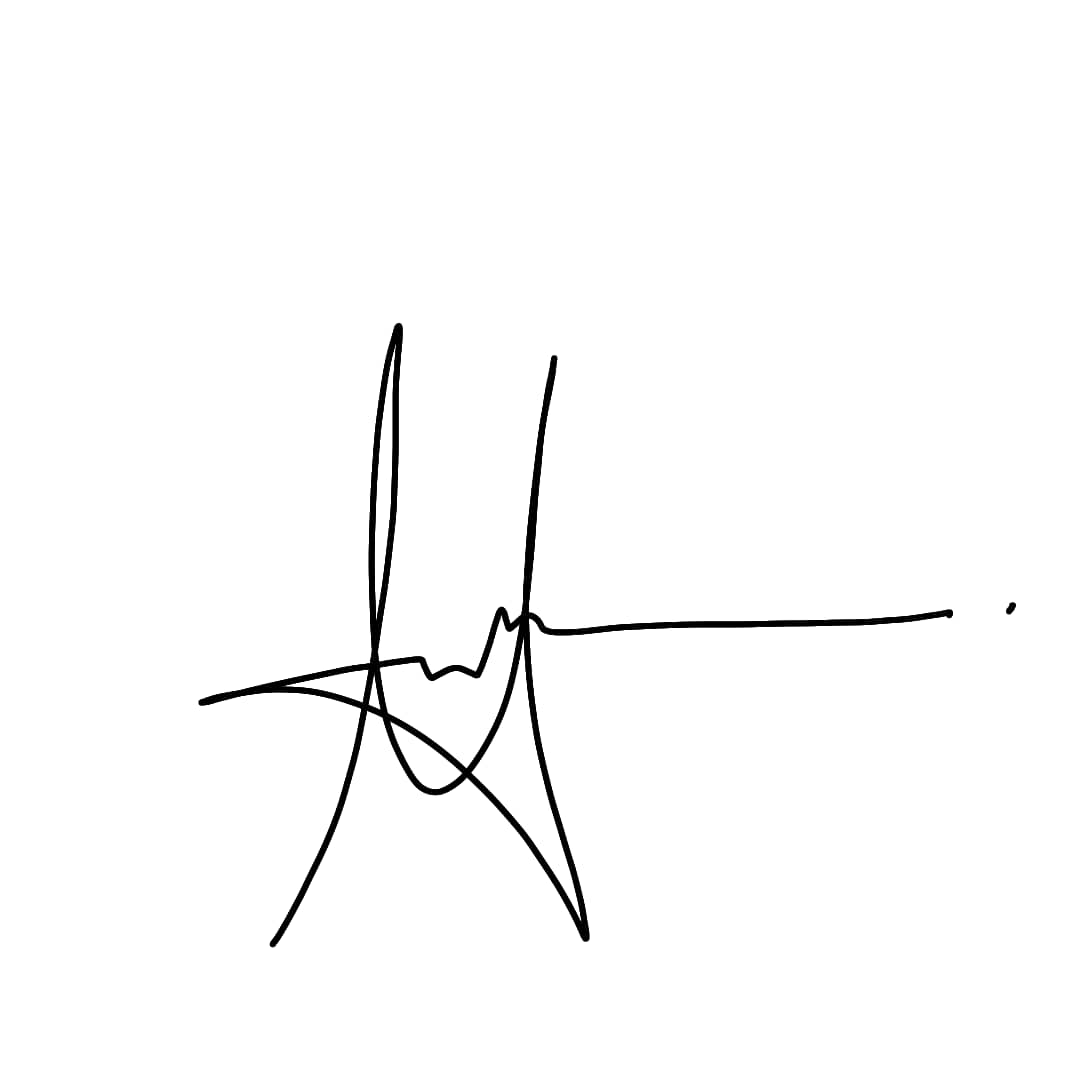
**Lecturer’s Remark** (Use additional sheet if required)

We.................................... (Name) ...................std. ID received the assignment and read the

comments....................................................................................................... (Signature/date)

**Academic Honesty Acknowledgement**

“We **Lee Pui Ling Elizabeth, Sasnitha Kandasamy, Nimue Wafiya, Hiba Azhari** verify that this paper contains entirely our own work. We have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, we have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. We realize the penalties *(refer to the student handbook diploma and undergraduate programme)* for any kind of copying or collaboration on any assignment.”



(Student’s signature / Date) **(17/11/2019)**

***Table Of Contents***

1.0 Introduction…………………………………………………………………….……3

2.0 Literature Review…………………………………………………………….…..4-6

3.0 Methodology

3.1 Binary Dilation……………………………………………………………….6-7

3.2 Vertical coordinates…………………………………………………………..8

3.3 Horizontal coordinates……………………………………………………………...9

4.0 Results and Discussion……………………………………………………………9

5.0 References………………………………………………………………………....10

***1.0 Introduction***

The topic we have chosen is c) text localization and detection for scientific document. Text detection and classification in documents is an essential role in finding the textual information for several computer vision applications such as optical character recognition, distinguishing between human and machine inputs and spam removal.

In computer vision, the localization of text in a scanned image of any text document is the first step in *Document Layout Analysis* [1]. Document Layout Analysis involves identifying and categorizing regions of interest in a scanned image of a text document, to then translate them into machine code and eventually, actual text (as opposed to just text inside the image). Before the image with text is sent through a reading system to be translated, it must undergo a process to be prepared to be accepted by that system. That system is what we will be attempting to program. Our aim is to locate blocks of text by segmenting them from non-textual zones and differentiating one text block from another. In other words, to identify each paragraph on a page.

In this report, we will first introduce three algorithms used for text localization and discuss the benefits and drawbacks of each. Then we will document our developed algorithm step-by-step and mention the author of each segment of code. Finally, we will discuss the results of our algorithm and compare it to the pre-existing algorithms mentioned above.

***2.0 Literature Review***

There are several methods for localization and extraction of text regions within a document. Here we discuss three common algorithms.

1. *Document Analysis System (1982) [1]*

This approach locates text in a document by first applying an algorithm to extract all foreground components into page segments, with each segment containing only one data format (text, image, etc.). Different features of these segments are then analysed using a classification process to determine the blocks containing text.

In the first step, extracting blocks of foreground components is done using a run-length smoothing algorithm (RLSA). This algorithm works on an image of the document where white (background) areas are represented by 0s and black (foreground) areas by 1s. The algorithm takes a value “C” and, while running through all the document image bits, converts 0s to 1s if they have less than C number of adjacent 0s. As explained in [1], the following string of bits will be converted, by the algorithm, as shown in Fig.1.

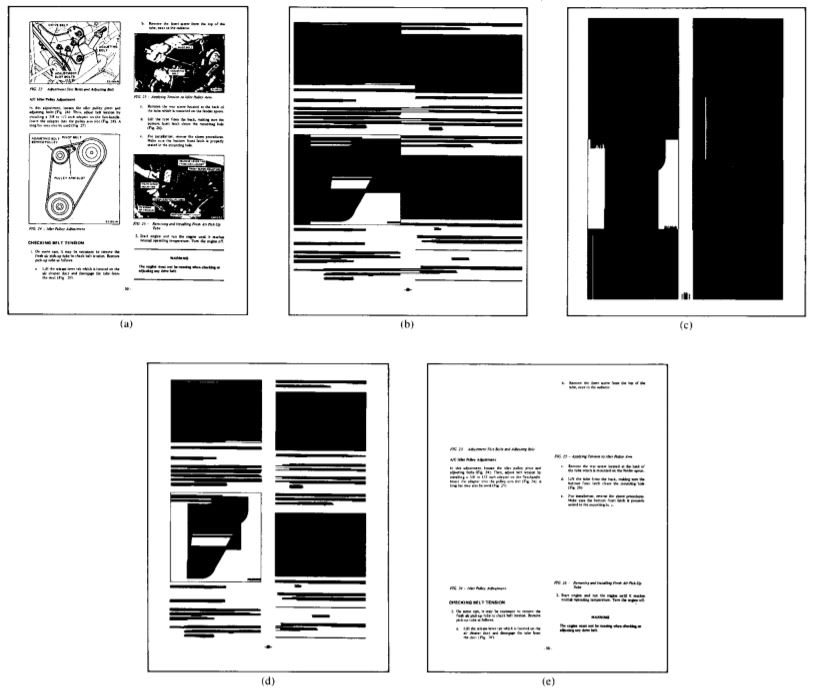
00010000010100001000000011000

Becomes:

11110000011111111000000011111

**Fig.1**

Therefore, neighbouring black areas separated by less than C pixels will be grouped together forming a block. The RLSA algorithm is run in horizontal and vertical directions and the output of both is combined to produce the final mask containing document blocks. This is demonstrated in Fig.2 retrieved from [1].



**Fig.2**

Features of the block, such as the height of each block segment and the mean horizontal length of blocks, are then calculated to differentiate between text and image blocks.

While this method is sufficiently effective, it is still prone to misclassification between text and image objects as images can exhibit features that are similar to those of text or contain text within them.

1. *Pixel labeling (2013) [2]*

This approach aims to classify pixels belonging to different document elements and extracting text pixels. Textual areas are regarded as textured while non-text contents are regarded as “regions with distinct textures” [2]. It simply separates different foreground components of the pages from the background based on variation in texture.

According to [2], this method is composed of two main stages. Stage one is selecting a random group of foreground pixels from a few pages of a book and calculating their autocorrelation to locate clusters of homogenous regions in the book. The aim is to find regions with similar textural content. Stage two is the processing of these foreground pixels to extract information.

This approach involves a greater attention to detail. This is because it is mainly geared towards hand written and historical books, which may be more difficult to process due to inconsistency. For this reason, it is understandable that, unlike the first algorithm, this algorithm requires some prior information on the nature of the document being processed. However, it is capable of interpreting a greater range of document formats.

1. *XY-cut Algorithm (1984) [3]*

This algorithm follows a slightly different approach for document segmentation. It locates the vertical and horizontal white spaces in the document and uses them to divide the document into paragraphs, lines and words. This produces pages divided into rectangles containing objects or white space which can then be analyzed to extract text regions.

The algorithm follows a recursive process. Starting from a full page of the document, it makes cuts to divide the page, based on a specific set of rules, into smaller rectangles.The information on all the rectangles is stored in the form of an X-Y Tree, where the root of the tree is the rectangle representing the entire page and the leaves contain the page’s subdivisions.

After division is complete, lines of text can be recognised by finding sequences of adjacent character blocks of the same height. And from lines, locations of paragraphs can be derived.

This approach can be highly dependent on the parameters (rules) used to determine page segmentation. These may need to vary greatly from one application of the algorithm to another.[5]

***3.0 Methodology***

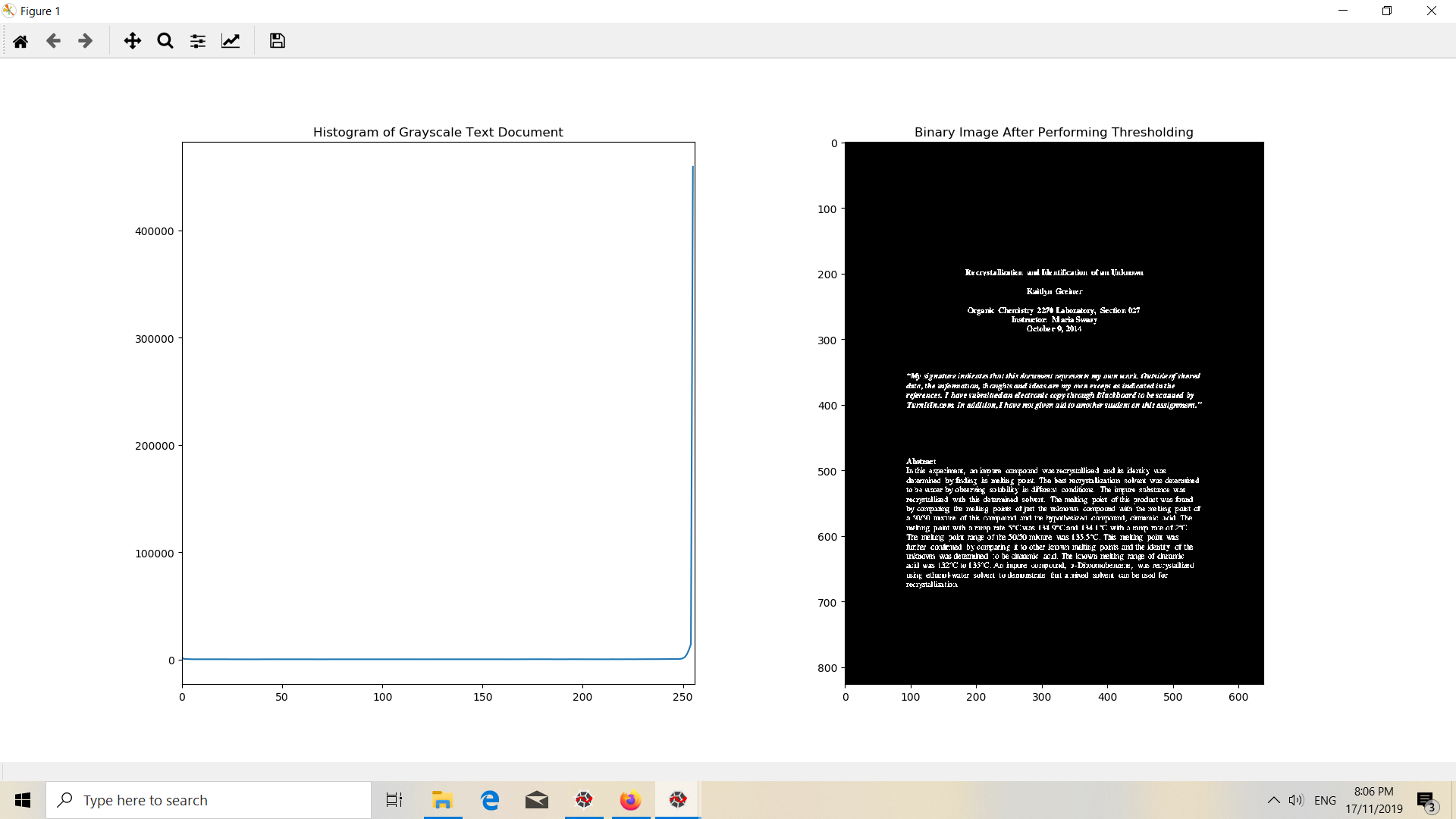
We have decided to develop our own algorithm to locate text within a document. Here we present a simpler approach with relatively limited capabilities. Our approach involves two main stages which are:

* Dilation of text regions into a continuous blocks.
* Determining the location of the block within the document.

These steps are discussed in depth below.

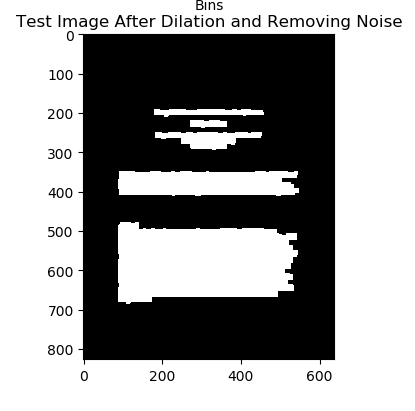
3.1 Binary Dilation

Firstly, the test image will be read as a grayscale image. Then, by using thresholding, it will be converted to a binary image and stored in another variable (bi\_img). The foreground objects and background are (255) and (0) respectively. It is important for the foreground objects to be white as we will be performing morphological operations on the binary image. ***{Elizabeth}***

**Fig. 3**   **Fig. 4**

As seen in the histogram in Fig. 3, the threshold value should be somewhere before the highest peak as that peak represents the white paper. To be on the safe side, we have decided to use a generous threshold value of 200. Based on the histogram the peak on the left represents the text whereas the peak on the right represents the background. Therefore, to perform the morphological operation on the binary image; the foreground object (text) has to be changed to white and the background to black by iterating through a for loop. The for loop is used by setting gray intensity levels that are less than the threshold value (the black text) to ‘255’ and gray intensity levels that are more than the threshold value to ‘0’. After that, to dilate the text, a morphological operation known as binary dilation is performed on the binary image to ease the process of detecting the text. Dilation is chosen because it will make the text more visible. This works as dilation enlarges the target via a ‘hit’ approach, thus, making the text grow and “bleed” into each other. After that, a closing operation is done to fill in small holes that are still found in the middle of the text blocks. The diagram (Fig. 5) shown below is the text image after dilation.

**Fig. 5**

3.2 Coordinate Location

Now that our text blocks have been bled together to form a large chunk of white pixels, we can begin iterating through the entire image to find the coordinates for each chunk. The aim is to get the y coordinates of the top and bottom of the chunk and the x coordinates of the leftmost and rightmost points of the chunk. This is done in two steps: y-coordinates and x-coordinates.

The first step we are taking is to get the y-coordinate of the top and bottom of the text block. If we think about how iterative statements typically run from left to right and top to bottom, it can be easily said that the first y-coordinate we aim to find is the one for the top of the image. We took note that it will not necessarily be the first white pixel we encounter but could instead, be somewhere in the middle of the text column-wise as some paragraphs start indented and thus, the highest point of the paragraph is not at the far left of the block of text. Once the top y-coordinate (the row value) is found, we will store it in the variable *top*. ***{Sasnitha}***

The next step is to find the bottom y-coordinate. To do this, we will start scanning the document from left to right starting from the row below the top row. We will also need to introduce a new variable, *c*, which counts how many continuous rows of all black columns we encounter. *c* will increase every time a row full of black pixels only is detected. We have decided that if the value of *c* reaches 5 as we iterate through the rows, or in other words, if there are 5 continuous rows of all black columns, then the bottom of text is that particular row value that c == 5 but minus 5. Thus, the *bottom* variable that indicates the y-coordinate of the bottom of the text will store the value of row-5. ***{Nimue}***

At this point, we have decided to create a function called getCoordinates which accepts *top* and *bottom* as the arguments. Within this function the top and bottom values of a block are used to search for the leftmost and rightmost points in the block. This is done by looping through pixel rows and, for each row, determining the positions of the first and last white pixel and whether it is the furthest to the left/right so far. The function returns the block’s boundaries.

***{Hiba}***

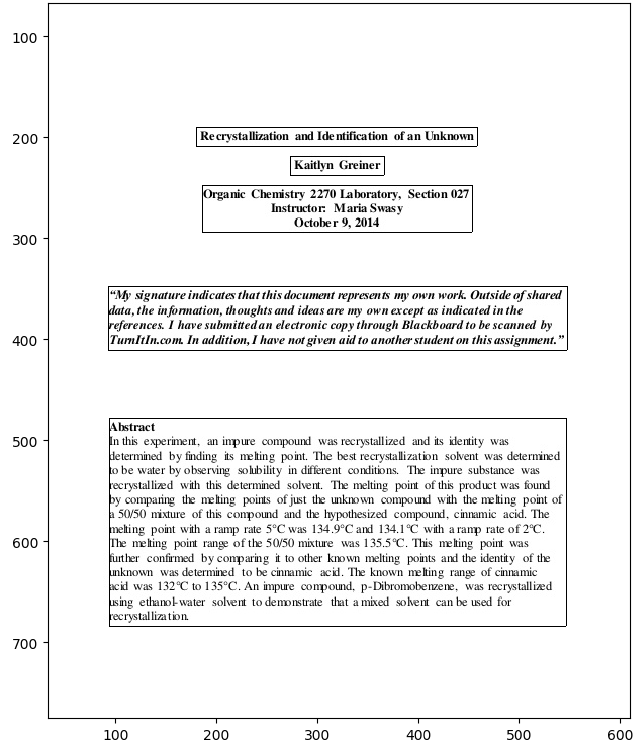
An array called coordinate\_list is created to store the coordinates for all text blocks. One item in coordinate\_list looks like: [top,bot,left,right]. This coordinate list would then be used to draw a rectangle around the located text.

It is important that after adding the coordinates for the block of text to coordinate\_list, the main loop will continue to look for additional texts of block beneath that block. So, the values for *top*, *bottom* and *c* must be flushed out or in other words, return to 0 so they may continue to store the new values for the next block of text. ***{Nimue&Sasnitha}***

To develop this, we discussed the algorithm then assigned different parts to each team member to be coded.

***4.0 Results and Discussion***

The following (Fig.6) was the result of the proposed algorithm after adding code to draw rectangles around text areas based on the calculated coordinates:



**Fig.6**

We can recognise the following important points in our algorithm:

1. Amendment is necessary to allow detection in multi-column documents. With our current algorithm, we can expect that multicolumn documents will be treated as a single column with separation occurring horizontally only. This can be rectified by making amendments to the coordinate extraction functions.
2. Different resolutions of documents will require different “lim” values as they may contain far more or less pixels between characters or paragraphs. They may also require greater dilation for the text to properly mesh together. These two parameters (extent of dilation and lim value) must be tweaked to suit the document being processed. Otherwise, we expect that the algorithm may, for example, treat spaces between lines within a paragraph as the end of the paragraph in a high resolution document image.
3. Classification of chunks of text from tables and diagrams will require further enhancement of the algorithm. If we were to apply this algorithm to documents containing various data types and aim to do classification work, enhancements to the code must be made.

Our algorithm, at its current state presented above, is well suited to extract text regions from single-column document images.

***5.0 References***

[[1]](https://pdfs.semanticscholar.org/7abc/a302c74d2f5adfd323a28e26d40b019df2b5.pdf) Namboodiri, Anoop & Jain, Anil. (2007). Document Structure and Layout Analysis. 10.1007/978-1-84628-726-8\_2.

[2] [Wong, K.Y., Casey, R.G., Wahl, F.M.: Document analysis system. IBM journal of research and development 26(6), 647–656 (1982)](https://pdfs.semanticscholar.org/7abc/a302c74d2f5adfd323a28e26d40b019df2b5.pdf)

[[3] Mehri, M., Heroux, P., Gomez-Kr ´ amer, P., Boucher, A., Mullot, R.: ¨ A pixel labeling approach for historical digitized books. In: Document Analysis and Recognition (ICDAR), 2013 12th International Conference on. pp. 817–821. IEEE (2013)](https://hal.archives-ouvertes.fr/hal-00934585/document)

[[4] Nagy, G., Seth, S.: Hierarchical representation of optically scanned documents. In: Proceedings of International Conference on Pattern Recognition. vol. 1, pp. 347–349 (1984)](https://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1292&context=cseconfwork)