

Agent Based Optical Character Recognition

Author: Pelin Kocyigit

A dissertation Submitted to the degree of
MSc. Computer Systems
School of Computer Science, Bangor University
Sep, 2012

Supervisor: Dr. W. J. Teahan

Acknowledgements

First of all, I would like to thank Allah for providing me the blessings to complete this work.

I owe sincere thankfulness to my supervisor, Dr. William J. Teahan, for his valuable contributions, patience and guidance through this study. I would like to also express my gratitude to Prof. Ludmila Kuncheva for her help, advice and encouragement.

Also, I would like to express my special thanks to my husband, Fatih, who made me believe in myself and never hesitated to support me.

My study in Bangor University during 2011/2012 was made possible by a Turkish Government Scholarship.

Statement of originality & release

Statement of Originality

The work presented in this dissertation is entirely from the studies of the individual student, except where otherwise stated. Where derivations are presented and the origin of the work is either wholly or in part from other sources, then full reference is given to the original author. This work has not been presented previously for any degree, nor is it at present under consideration by any other degree awarding body.

Student:

.....

Statement of Availability

I hereby acknowledge the availability of any part of this dissertation for viewing, photocopying or incorporation into future studies, providing that full reference is given to the origins of any information contained herein.

Student:

.....

Company Representative (if applicable)

.....

Abstract

This dissertation presents a new approach for optical character recognition (OCR). The new approach extracts characters from an image by walking over them on an agent based environment. To this aim, an OCR system is developed in this study. The OCR system is implemented in Java, NetLogo and MySQL environments. Java creates the interface of the system and runs NetLogo. NetLogo is an agent based modeling system and provides the required environment to implement the new approach. Thus a model in NetLogo is proposed that determines characters on the image and derives some results by walking over them. The results of the walking movement over a character are matched against a dataset contained by MySQL and the most similar character which possesses this results is found from the dataset.

The proposed OCR system was tested on various images of the texts and performance of the system was also compared with an online OCR application. It was observed that the proposed system reached high recognition rates and introduced solutions for some common problems of OCR systems.

This dissertation also presents a new thinning algorithm which was developed to make the characters in the image suitable for the walking task of NetLogo. The new thinning algorithm reduces the size of the characters from many pixels to a single pixel width. The effectiveness of this algorithm was compared with some other popular thinning algorithms. It was determined that the new thinning method provided better results on small sized characters by protecting important points of the characters.

Contents

1	Overview	1
1.1	Introduction	1
1.2	The Idea	2
1.3	Aim and Objectives	4
1.4	Structure of Presentation	4
2	Background	5
2.1	Machine Learning	5
2.2	History of OCR	6
2.3	Use of OCR	7
2.4	OCR	7
2.4.1	Systems According To Text Type	7
2.4.2	Systems According To Data Collection	8
2.4.3	Methodologies of OCR	9
2.5	Evaluation of OCR Systems	17
2.6	A Quick Review of NetLogo	19
2.7	Summary	19
3	System Design	21
3.1	Structure of The System	21
3.2	NetLogo for OCR	23
3.2.1	The Advantages of NetLogo	24
3.3	Summary	26

4	Image Preprocessing	27
4.1	Image Processing	27
4.2	Convolution Filters	29
4.2.1	Sharpening Filter	29
4.2.2	Edge Detection Filter	30
4.2.3	Noise Removing	31
4.3	Thresholding Algorithms	32
4.4	Thinning Algorithms	33
4.5	A New Approach for Thinning	34
4.5.1	Methodology of The Algorithm	35
4.6	Summary	39
5	Agent-based OCR in NetLogo	40
5.1	Displaying the Processed Image in the NetLogo World	40
5.2	Methodology of the Agent-based Approach for OCR	41
5.2.1	Line Determination	42
5.2.2	Character Recognition	43
5.2.3	Character Matching from the Database	48
5.2.4	Displaying Results	53
5.3	Summary	55
6	Evaluation	56
6.1	Postprocessing of Textual Output	56
6.2	Evaluation of the Textual Output	57
6.3	Analysis of the Evaluation Results	58
6.4	Comparison with Other OCR Systems	59
6.5	Summary	61
7	Conclusions and Future Work	62
7.1	Summary of the Dissertation	62
7.2	Contributions	64
7.3	Discussion	65
7.4	Future Work	66
	Bibliography	72

Appendix A Pseudocode of Character Recognition	73
Appendix B UML Diagram	78

List of Figures

1.1	An example how the character “L” may be walked over.	3
1.2	An example of the NetLogo Interface.	3
2.1	An example of the neighborhood of a pixel.	10
2.2	An example of two convolution filters.	11
2.3	An example of the 3×3 Sobel Filter.	12
2.4	An example of the dilation operation to join broken segments of poor written characters [16].	13
2.5	The structuring element of Zhang-Suen thinning algorithm.	14
2.6	An example application of Zhang-Suen algorithm.	15
2.7	An example of the zoning.	16
2.8	An example of the crossing and distances methods [48].	16
3.1	A basic structure of the system.	22
3.2	The codes to find counts of neighboring pixels that are black in Java and NetLogo.	24
3.3	An example of cleaning spurious part of a character.	25
4.1	The first screen of the interface of the Java program.	28
4.2	The optional image processing methods.	29
4.3	An example of the effect of a sharpening filter.	30
4.4	The kernel of the sharpening filter.	30
4.5	An example result of the edge detection algorithm on a bold character.	30
4.6	An example of four edge detection filters.	31
4.7	An example of three noise removal filters.	31
4.8	An example of comparison of thinning algorithms on bold characters.	33
4.9	An example of comparison of thinning algorithms on different types of characters.	33

4.10	An example of two convolution filters.	34
4.11	An example of an image template and its array information.	36
4.12	An example of an image template.	36
4.13	The types of enumeration of neighbor pixels.	37
4.14	An example of pixels with two neighbors.	37
5.1	An example of transferring the image into the NetLogo world.	40
5.2	An example of transferred image to NetLogo world.	41
5.3	An example of edges.	44
5.4	An example of edges whose patches are connected as diagonal.	44
5.5	An example of corners which are based on Condition 1.	45
5.6	An example of corners which are based on Condition 2.	45
5.7	An example of a branch point which satisfies Condition 1.	46
5.8	An example of a branch point which satisfies Condition 2.	46
5.9	Example turtle movement strings and edge counts of some characters.	47
5.10	Edge comparison of character Y and v.	51
5.11	An example database query in NetLogo.	53
5.12	An example of the NetLogo model extracting the characters.	53
5.13	An example of the NetLogo model after the extraction of the characters completed. . .	54
5.14	Displaying the NetLogo model on the Java Interface.	54
6.1	An example of evaluation of recognized text	58
7.1	The process flow of the character recognition in the OCR implementation.	63
7.2	An example of spurious pixels between two characters.	66
7.3	An example of gap on the character.	66
A.1	Pseudocode of the Character Recognition Algorithm	77
B.1	UML Diagram for Java Classes	78

List of Tables

4.1	The rules of the new thinning algorithm.	38
5.1	Database of Gulim and Bitstream Vera Fonts.	48
5.2	The patch count of the edges for character Y and v.	51
6.1	An example of accuracy changing of a recognized text.	57
6.2	OCR implementation results of different texts in Gulim font type.	59
6.3	OCR implementation results of different texts in Bitstream Vera font type.	59
6.4	Accuracy comparison of some commercial OCR devices in 1992 [41].	60
6.5	Comparison results of our system with an OCR application at http://www.free-online-ocr.com web site.	60

Chapter 1

Overview

This chapter presents the subject and focus of this project. In the first section, an overview of character recognition systems are introduced. The second section introduces the aim of the project. In the third section, the objectives of the project are discussed, while the last section describes the presentation of the report.

1.1 Introduction

After the advent of digital computers, tailoring human functions to computers has been an interesting and exciting research field [6]. To enable the machine to recognize characters, efficient algorithms have been developed so far [14]. Optical character recognition (OCR) is one of the best outcomes of this research. OCR is a system developed for deriving character-based file from scanned images of handwritten, typewritten or printed text documents. In other words, it is simply a visual recognition process that converts text documents into editable texts.

OCR technology can significantly increase efficiency of office work. Instead of retyping documents, applying OCR technology to the image of the documents can save huge amounts of time. Based on this main advantage of OCR systems, today they are used in a wide area from archeology to banking.

There are two branches of character recognition depending on the input device: on-line and off-line recognition. In on-line recognition, data is acquired in real-time such as digitizer tablets. Off-line recognition systems collect data from static devices such as scanners and cameras. On-line recognition systems allow to the writing of information in real-time due to the concurrent data collection structure. In contrast, off-line recognition systems require specific methodologies to prepare the image for recognition process and to eliminate damage (noise and errors) of the input image caused

by collection process.

Recognition of machine-printed characters and clear handwriting has almost been achieved and there are commercial systems in the market. However, recognition of unconstrained handwriting characters and determining similarity between some characters, such as “o” and “0”, “p” and “q”, are still challenging problems. Although there has been substantial research in character recognition, there are no satisfactory solutions for these problems of OCR systems.

In this project, we have focused on experimenting with character recognition in a new platform, NetLogo, and developing a new approach to solve character matching problems. Working with NetLogo provides an agent oriented environment. By controlling the agents, the recognition process can be observed simultaneously and specific calculations for each character are conducted using the attributes of the agents. This major advantage of NetLogo can present more robust solutions in character recognition.

1.2 The Idea

The accuracy of character recognition systems depend on the feature extraction of characters. To recognize a character from the other characters, each character needs to have specific distinguishing features. These distinctive features are based on the written type of the characters. To write a character ourselves, we choose the best way of “walking” over the character by following its shape. (Imagine a robotic turtle walking over a giant character. The idea here is using a virtual robotic turtle in NetLogo to perform the same task.) Namely, when we write a character ourselves, we start at a certain position and move forwards in different directions to complete a character. For example, assume the start position of a character is the right bottom corner of the character. According to this start position, to write character “I”, we start from the start position and move forwards in a direction to north. And to write character “L”, we start from the start position and move forward to the west, then turn right and forward to north as shown in Figure 1.1.

So once we complete the “walking” movements on a character in order to write the character, we end up with specific characteristic features of the character which are the finish location of the character, the count of the steps taken, the directions used to follow the lines of the character and the order of these directions. These results are specific to each character, except for a few characters that are similar to each other, and give consistent information about characters. In this project, the features of characters for a particular font type are collected in a MySQL database to match the walking movement commands against a character in the database. It is shown that this approach can

increase the accuracy of OCR process and reduce the misrecognition errors that other systems make.

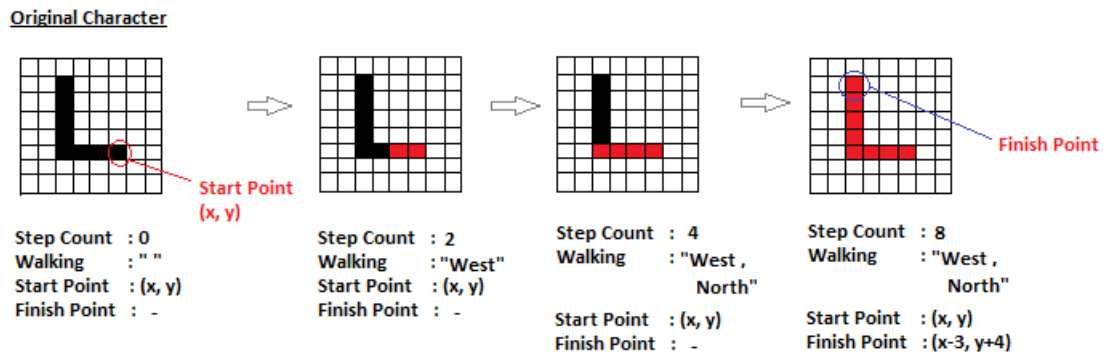


Figure 1.1: An example how the character “L” may be walked over.

The most important part of the project is the walking movements performed over the characters. To fulfill this requirement, the NetLogo programming language is picked due to the agent based environment and an agent of the NetLogo is charged for the walking task as shown in Figure 1.2. NetLogo presents an environment to display the characters and easily allows the walking movements of the agents to be specified using intuitive first-person commands such as forward, left and right. Also, some image processing algorithms are required to display the characters in NetLogo environment. Thus Java programming language will be used to implement the image processing algorithms.

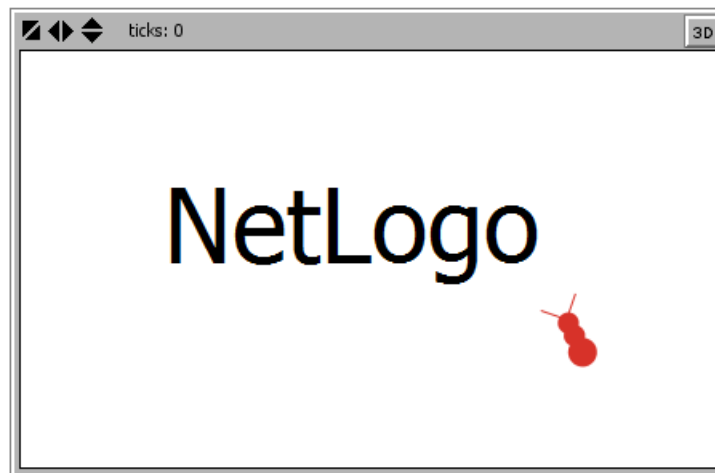


Figure 1.2: An example of the NetLogo Interface.

1.3 Aim and Objectives

The overall aim of this project is to assess the effectiveness of a new OCR algorithm that uses an agent based approach as outlined in the previous section. To this aim, the specific objectives of this project are:

- To implement the image processing step of OCR, possibly developing improved algorithms in the process.
- To explore methods to communicate between Java and NetLogo.
- To implement the character recognition process in NetLogo using agent based approach.
- To create a specific database of features of characters for a particular font type and to provide access to MySQL databases from NetLogo that can be used to match characters with the new algorithm.
- To evaluate the effectiveness of the final OCR implementation.

1.4 Structure of Presentation

This project report consists of seven chapters.

This chapter introduces the subject and presents the motivation behind the project. Chapter 2 provides detailed background work and methodologies related to the research subject. In Chapter 3, the structure of the program developed in this project and required algorithms to fulfill the objectives of the project are explained. Chapter 4 introduces the algorithms used for image processing in Java. Chapter 5 explains the new approach for character recognition and introduces the model developed in NetLogo for the recognition task. In Chapter 6, evaluation of the proposed OCR system is described. The last chapter discusses the contribution of the project, and how the new character recognition approach can be improved in the future.

Chapter 2

Background

This chapter discusses background of the OCR technology and some specific methods of character recognition that are adopted in this project. The first section explores machine learning and second section discusses background studies of the OCR systems. In the third section, application of OCR technology is introduced. Section four explains the process flow of the OCR systems and the algorithms used for character recognition. Section five presents the methods to measure performance of the OCR systems while the last section gives a brief introduction to the NetLogo environment.

2.1 Machine Learning

”Find a bug in a program, and fix it, and the program will work today. Show the program how to find and fix a bug, and the program will work forever.”- Oliver G. Selfridge, in AI’s Greatest Trends and Controversies.

Machine learning(ML) is a branch of artificial intelligence which is concerned with development of computer algorithms that learn [44]. Machines can learn and perform a wide range of tasks [33]. Over the past two decades, ML has become one of the most important technologies and used in various scientific domains such as computer vision, recognition, robotics, optimization and theoretical computer science [34].

How a machine can learn a task and perform the task are depends on the learning task and learning type. There are two major learning types, namely, supervised and unsupervised learning. Supervised learning generally attempts to get the computer to learn a predefined classification technique such as handwriting recognition. The performance of the handwriting recognition is measured by the rate of

words correctly classified. However, there is no classification provided in unsupervised learning type. In fact, the main task of it is to develop classifications seeking out similarity inside the data such as data mining clustering algorithms.

2.2 History of OCR

The first attempts in character recognition area were in 1870 with the creation of the retina scanner. C.R.Carey of Boston Massachusetts invented this device as an image transmission system with the use of a photocell mosaics. After a while, in 1890's, the invention of the sequential scanner by Paul Nipkow made a real breakthrough in machine reading. He used an optical mechanical picture scanning method which divides an image into line and point mosaics [45].

The engineering attempts at character recognition and creation of the modern version of OCR started after the development of digital computers in 1940. But it was for very limited data processing and specially designed character sets within the business world. So, fixed location and form were required in documents for the recognition process [15]. By 1950, electronic data processing was becoming popular research field and the first OCR device was introduced in 1954. Recognition of typewritten characters was achieved with this machine and OCR machines were commercially ready in the market. But, due to the high cost of the hardware systems, OCR systems could not fulfill expectancy.

In the 1960's, a method called curve following was developed to provide flexibility in character design using a cathode ray tube and photo multiplier. This was the first step that made the idea of recognition of handwriting possible by manipulating data [15].

After recognition of typewritten characters, handwriting recognition was accomplished in 1970's. Although character sets were still limited, it was a remarkable progress to improve OCR technology. Also, in this period IBM, Toshiba and Hitachi made some work to increase the character sets and performance, and to reduce the cost.

OCR technology had started to appear in commercial life in the early 1970's with the invention of a system to read credit card receipts. After a decade, a wide variety of applications from passport processing to food packaging have discovered the noteworthy abilities of character recognition systems. The first passport scanner of the U.S. was introduced by Caere Corporation in 1984 [21].

After this period, ongoing research in the machine reading area was expanded in parallel with the necessity of new products [9]. As hardware was getting cheaper, and OCR started to be developed as software systems, popularity of this technology increased [11]. Today, there are some OCR appli-

cations whose accuracy are 99 % for recognition of type written documents and a few thousand OCR systems are sold every week [11].

2.3 Use of OCR

The first application of this technology was used in banking. Today, OCR is still an important technology in banking. It helps to read handwritten characters and numbers on checks so that the requested money is easily transferred without human control. It saves time to both customers and banks, and emphasizes the importance of accuracy of OCR.

OCR appeals to applications that need to work with documents. Such applications can be found in industrial firms and hospitals. In firms, documents contain bill, product, customer and staff information. Instead of recording the text manually, using OCR can save time and also reduces the possible syntax errors on manual typing. Likewise, the obvious application of OCR systems in hospitals is for patient forms or other related documents. Students and scientific researchers can use OCR systems for extracting text from scanned images. They can just take the image of the interested part from books, magazines and can make use of them in machine-coded format rather than having to type in.

Thanks to the combination of OCR systems and text-to-speech technology, applications are developed to read documents for blind people. OCR is useful in robotics too. With a camera and an embedded OCR software application, robotic eyes can read. One of the exciting usages of OCR is for historical documents and scripts. Archivists benefit from machine reading to convert massive quantities of handwritten historical documents into digital form.

2.4 OCR

While reading of typewritten characters is accurately achieved, OCR systems still can be inaccurate for handwritten characters. This makes character recognition still a challenging and open area for research [16]. OCR systems can be categorized based on text type and collection of the data.

2.4.1 Systems According To Text Type

There are two main text types in OCR: typewritten characters and handwritten characters. The major advantage of typewritten characters is the limitation of the possible text styles. This reduces the processing time and increases the possible accuracy rate.

Recognition of Typewritten Characters

Typewritten (printed) characters have a fixed size, measurable direction and relatively clear font style [24]. In the recognition process, a possible slant is estimated, and the image of the document is divided into lines, words and characters. Constrained writing style and regular letter separation in printed documents increase the performance of the application. Today, the available commercial OCR systems can reach 99 % accuracy for typewritten recognition [31].

Recognition of Handwritten Characters

Despite the growing interest in character recognition and a wide range of applications, there still isn't any satisfactory solution for the handwriting recognition task [23]. The main reason lies behind the fact that there are inconsistent writing styles.

There are some common problems in different approaches to analyze handwritten characters. Some of the problems are :

1. Similarity of The Characters: Recognizing differences between some characters is quite complicated and requires specific algorithms. Examples of such characters are “o” and “0”, “I” and “1”, “g” and “q”.
2. Handwriting Types: Inconsistency of size, trend, spurious parts and distortions of letters affect the accuracy rate.
3. Paper Quality: Background of the text is one of the important factors in OCR. Colored, noisy and low quality papers decrease the likelihood of finding the true letter.
4. Different Alphabets: Some alphabets such as Cyrillic, Latin, Chinese have a character set which is difficult to analyze. The Chinese national character set consists of 27484 characters and OCR applications are unable to correctly separate all symbols.

2.4.2 Systems According To Data Collection

The character recognition task can be examined according to two data acquisition methods: Off-Line Recognition and On-Line Recognition. The mode of data acquisition is determined by the input devices. In off-line recognition, these devices are static devices such as scanners and cameras. In on-line recognition, dynamic devices are used such as graphic tablets and digitizing tablets. Dynamic devices benefit from being able to observe the process of writing concurrently.

On-Line Recognition

Recognition of on-line characters requires an immediate interaction with the user and a specific equipment to manage the application [30]. It works in real-time and so allows the capturing pen movements (pen-up/pen-down) to record information related to writing. Writing order, velocity and acceleration of the pen help clarify the process of understanding the character. Also, stroke detection can be analyzed following pen movements [29].

Dynamic recognition requires less processing than static recognition because some steps are already done while recording writing information. This advantage of on-line recognition simplifies reading characters and provides better recognition accuracy [30].

Off-Line Recognition

This is also known as OCR and does not require an immediate interaction with user. Data input is accessed via static devices. On the one hand, this allows for analyzing the current character in the context of previously written characters. But on the other hand, there could be noise or blur on the image and so more processing steps are required [19].

The process of off-line recognition is not real time. Thus it does not contain any information about the order of the characters and direction of the lines. This disadvantage of the off-line recognition process requires more costly and longer processing time.

2.4.3 Methodologies of OCR

The OCR process follows the steps below [11]:

1. Preprocessing.
2. Segmentation.
3. Feature Extraction.
4. Classification and Recognition.

Preprocessing

In the preprocessing step, the irrelevant information in the image is removed and useful features of the textual parts are retained. To this aim, the image undergoes a set of preprocessing operations. These operations can be chosen according to the structure of the image. Some of the operations may be omitted or applied in a different order [22] [35]. The main methods performed in this phase are :

1. Gray Scale and Thresholding.
2. Color Image Processing.
3. Morphological Operations.

Gray Scale and Thresholding In a gray scale (monochromatic) image, each pixel carries only gray level intensity. To extract foreground and background pixels, the image is converted into a gray scale format.

Thresholding is the step used to convert gray scale images into binary images. There are two main thresholding methods: global and adaptive. The global thresholding method uses a constant threshold value and the intensity of each pixel is compared with this threshold. If the pixel's intensity is greater than the threshold, this pixel is marked as a foreground pixel. To find the optimum threshold, the Otsu thresholding method has been used in a wide range of applications [8] [38].

Adaptive thresholding is suitable for images that require different thresholding values for particular regions of the image. Among the different methods, Niblock's method produces the most appropriate thresholding values [37].

Color Image Processing The methods of color image processing are mostly based on image filtering. Image filtering is the process of modifying the pixels in an image to create different effects. Image filtering techniques use the neighborhood of any given pixel and image convolution filters. A pixel's neighborhood is composed of a series of pixels that are connected to the given pixel as shown in Figure 2.1.

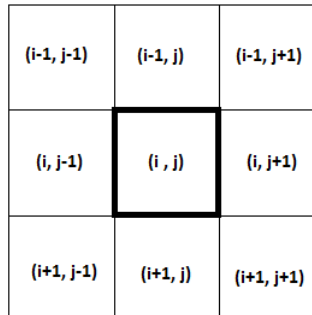


Figure 2.1: An example of the neighborhood of a pixel.

The convolution operation is the process of multiplying color values of a selected pixel and its neighbors by using convolution filters [32]. The convolution filters, also known as filter kernels, are

$n \times m$ matrices where n and m are the width and the height of the filter [32]. Mathematically, the convolution operation is defined as :

$$I'[u, v] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(u-i, v-j) H(i, j) = I \otimes H, \quad (2.1)$$

where $I(u, v)$ is the original image, u and v are the dimensions of the image, $I'[u, v]$ is the filtered image and $H(i, j)$ is the filter kernel. Convolution filters are mostly used for removing noise, deblurring and edge detection operations.

Noise Removal: In electronic images, noise can occur due to the transmission of data. To remove this type of noise, the gray level image is smoothed by using linear filters and non-linear filters [16].

In linear filtering, weighted averages of the input values are computed to smooth the image. Some of the common linear filters are Mean (averaging) and Gaussian Filters. The idea behind smoothing is to replace the color value of a given pixel by the average of its neighbors [16]. The mean filter is one of the simplest ways of smoothing, as shown in Figure 2.2. On the other hand, it can cause a reduction in the sharpness of the image by blurring the edges.

			1	4	7	4	1
			4	16	26	16	4
			7	26	41	26	7
			1	4	7	4	1
			4	16	26	16	4
1	1	1					
1	1	1					
1	1	1					

(a) Mean
(b) Gaussian

Figure 2.2: An example of two convolution filters.

Non-linear filtering is more effective than linear filtering at preserving the useful details of the image such as a Median Filter. Usage of the Median Filter is similar to the Mean Filter but the median of the input values is calculated rather than the mean.

Deblurring: Blurring can occur due to movement of the objects while taking a picture [7]. The image is deblurred using Inverse and Wiener Filters. The inverse filtering is the process of recovering the image for deconvolution. It is very sensitive to noise whereas Wiener filtering overcomes this

limitation. The Wiener filter is a statistical method proposed by Norbert Wiener [49]. It removes the additive noise and inverts the blurred image concurrently.

Edge Detection: Edges (contours) in the image correspond to the boundaries between two regions [27]. Edges can be determined using the changes in color discontinuities. The intensity of the color is generally changed in gradient points. The gradient of an image is calculated by the Formula [42]:

$$\nabla f = \left[\frac{\partial f}{\partial x} \hat{x}, \frac{\partial f}{\partial y} \hat{y} \right], \quad (2.2)$$

where $\frac{\partial f}{\partial x}$ is the gradient in the x direction and $\frac{\partial f}{\partial y}$ is the gradient in the y direction. Most of the edge detection filters, such as Roberts, Prewit and Sobel, are based on gradient calculation methods [12].

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

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

(a) Sobel Filter in x direction. (b) Sobel Filter in y direction.

Figure 2.3: An example of the 3×3 Sobel Filter.

Morphological Image Processing Morphological operations are the techniques which deal with extraction of useful image components such as skeletons or the convex hull [16]. Morphology presents a set of robust approaches to various image processing problems. One of the approaches is the Minkowski method. Minkowski morphological operations are based on replacing convolution by logical operations [39] [43]. They use a structuring element to process the image. The structuring element is a pattern used to interact with the image to draw results on how the pattern fits or misses the region in the image. They are used to smooth boundaries of the region and reduce noise and artifacts.

Some of the major morphological operations are dilation, erosion and skeletonisation.

Dilation: The dilation operation is applied to binary images to enlarge the boundaries of the foreground pixels of a region. The dilation operation of A (a binary image) by B (a structuring element) is :

$$A \oplus B = \bigcup_{b \in B} A_b, \quad (2.3)$$

where A_b is the translation of A located at the origin of b . It is mostly used for bridging gaps as shown in Figure 2.4 [16]. The origin of the structuring element is compared with each pixel in the image. If at least one item of the structuring element matches with a foreground pixel among the input pixels (neighbors of the selected pixel), the selected pixel is set to the foreground value. If all items of the structuring element matches with background pixel, the pixel is set to the background value.

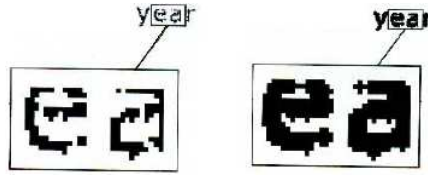


Figure 2.4: An example of the dilation operation to join broken segments of poor written characters [16].

Erosion: The erosion operation is to erode away the boundaries of the foreground pixels. Thus the size of the foreground pixels is reduced whereas holes inside this region are expanded. The formula of erosion operation is:

$$A \ominus B = \bigcap_{b \in B} A_{-b}. \quad (2.4)$$

This method is applied as dilation but with a different algorithm. If all items of the structuring element match with a foreground pixel, the selected pixel is left as the original. If any item of the structuring element matches with a background pixel, then the pixel is set to the background value.

Skeletonisation and Thinning: Skeletonisation is the process of reducing the width of a region to just a single pixel. Like the dilation and erosion operators, a structuring element is used for skeletonisation methods. This operator provides extraction of critical regions such as connection or junction parts. There are two major categories in skeletonisation operations: Sequential and Parallel. Both of them use neighborhood and connectivity attributes of the pixels.

The Medial Axis Function (MAF) was the first definition of the skeleton by Blum in 1967 [36]. MAF is the group of points with more than one neighbor in the boundary of the region. The skeleton

of the image is defined using MAF and thinning algorithms.

Thinning is an essential method of the skeletonisation operation [52]. It works iteratively for the regions of straight and curved lines. Zhang-Suen, Holt and SUSAN are some of the effective and popular thinning algorithms. Zhang-Suen algorithm uses a 3×3 structuring element and consists of two subiterations [51]. There are 9 pixels in the structuring element and they are enumerated in a clockwise manner as shown in Figure 2.5.

P₉ (i-1, j-1)	P₂ (i-1, j)	P₃ (i-1, j+1)
P₈ (i, j-1)	P₁ (i, j)	P₄ (i, j+1)
P₇ (i+1, j-1)	P₆ (i+1, j)	P₅ (i+1, j+1)

Figure 2.5: The structuring element of Zhang-Suen thinning algorithm.

Both subiterations of the Zhang-Suen algorithm consist of four rules. The rules are applied if the central pixel of the structuring element is a foreground pixel. If all conditions are satisfied, P_1 (central pixel of the structuring element) is marked to be deleted. Both iterations are repeated until there is no pixel to delete. These are the rules of the first iteration:

1. The connectivity number of the central pixel is one. (Connectivity is the number of 01 patterns in the ordered set $P_2, P_3, P_4, \dots, P_9$)
2. The central pixel has at least two foreground neighbors and not more than six.
3. At least one of P_2, P_4 or P_6 are background pixels.
4. At least one of P_4, P_6 or P_8 are background pixels.

All marked pixels are deleted at the end of each iteration. After the first iteration is completed, the second iteration is applied. The second iteration applies the same rules as the first iteration, except for rule 3, where P_6 is replaced by P_8 and rule 4, where P_4 is replaced by P_2 . With the help of this algorithm, all contour pixels except those belonging to the skeleton are removed.

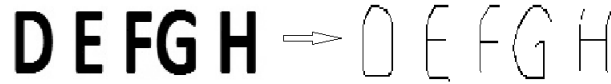


Figure 2.6: An example application of Zhang-Suen algorithm.

Segmentation

The aim of this phase is segmenting text lines of the image into sentences, words and characters [22]. Segmentation is an important step because extraction of the characters in this stage directly affects the accuracy of the recognition process. There are two main types of segmentation: Explicit and Implicit.

The explicit method is to determine explicitly segmentation points in a word by using some particular points such as spaces or intersection points [3]. The explicit method analyzes the page layout into its logical parts in two steps. The first step is responsible for segmenting the text lines as paragraph, rows and words. In the second step, the functional components of the document are extracted based on the location of the points.

In the implicit method, segmentation and recognition of characters are managed sequentially [40]. It searches the components of the image which match classes in its alphabet. So, all decisions for segmentation are postponed until the recognition phase [46]. There are some problems in this method. For example, a part of a character can be divided as a segment of another character. In [5], some loss in the recognition performance is observed, caused by concurrent recognition segmentation.

Feature Extraction

This phase extracts the most representative information from the raw data before the recognition process. A well-defined feature set can distinguish the characteristics of a class from those of other classes. Various feature extraction techniques are grouped into three main categories [10] [8]: Geometric Features, Structural (Statistical) Features and Global Transformation.

Geometric Features The geometric features contain moments, histograms and direction attributes [22].

Moments: They are global properties of the characters such as the shape of the region or the mass center [11].

Histogram: They represent the distribution of pixels in gray scale images [8].

Direction Features: Characters consist of strokes which are connected as lines, curves or polylines. The direction of strokes is an important factor to reveal symbolic information of characters among hundreds of pixels. The local direction of a character can be determined by different methods such as skeleton structure, stroke segment and gradient direction.

Statistical Features These are determined using statical distribution of points. Some of them are introduced below.

Zoning: The character is contained in a frame. The frame is divided into zones according to the densities of points as shown in Figure 2.7. Thus the density of each zone is calculated and used in feature determination [11].

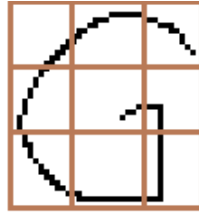


Figure 2.7: An example of the zoning.

Crossing and Distances: The crossing technique counts the number of transitions from background to foreground pixels through horizontal and vertical lines of a character. Distance techniques count the distance from the first boundary pixel of the character to boundary of the zoning frame as shown in Figure 2.8.

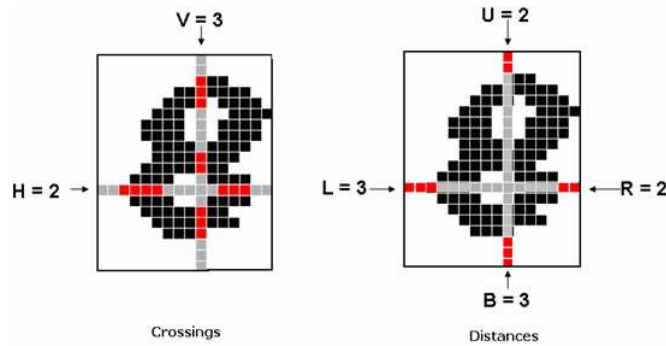


Figure 2.8: An example of the crossing and distances methods [48].

Global Transformation This methods reduce the feature vector's dimensionality. They generally use the cursive parts of the characters that are the letters in a word joined together. So they are very sensitive to gaps over the edges of cursive character. Haar, Fourier and Hough are some of the popular transformation methods [1].

Classification and Recognition

In the last step of the character recognition task, characters are classified and recognized according to their features. The main approaches can be adopted in this level [2]:

1. Template Matching.
2. Statistical Techniques.
3. Structural Techniques.

Template Matching It is the simplest way of character recognition by matching predefined character prototypes against the character to be recognized [6]. It classifies a character by determining its similarity to the templates.

Statistical Techniques These are statistical decision functions and a group of optimality criteria. They are based on three assumptions [10]:

- Distribution of the feature set is Gaussian or in the worst uniform.
- There are sufficient statistics available for each class.
- Given ensemble of images I , one is able to extract a set of features $f_i \in F$, $i \in 1, \dots, n$ which represents each distinct class of patterns.

Structural Techniques The structural techniques create syntactic patterns to measure similarity between structural components [19]. Each class has its own grammar to describe the structure of the character. Structure components extracted from the character to be recognized are matched against this grammar class.

2.5 Evaluation of OCR Systems

Performance of the OCR systems vary from text to text depending on a number of factors. Some of them are:

- Paper quality (good quality papers or old papers).
- Scanning process of the texts.
- Written type of the texts (handwritten or typewritten).
- Font type and size.

These factors can lead to blur, noise and low resolution on the paper, can warp the characters at the edge of the text and create spurious parts on the characters [50]. OCR systems, which have adopted some specific algorithms (such as noise removing, deblurring) to remove the effects of these factors, can reach higher performance. To measure performance of an OCR system, the accuracy rate of the recognized text and time to perform recognition are used.

The Levenshtein method [25], also known as the edit distance method, is a popular algorithm to calculate accuracy rate of an OCR system. The Levenshtein method is also used for spell checking, DNA analysis and plagiarism detection. This method calculates a distance measure between two strings. The distance is the number of edits (insertion, deletion and substitutions) required to match a source string against a target string.

For example, the distance between “cool” (source string) and “coal” (target string) is 1, because one substitution (substitute “o” with “a”) is required to match the strings. The distance between “read” (source) and “ready” (target) is 1 due to requirement of the insertion of “y” into the source string. For the strings “blurry” (source) and “blur” (target), the distance is 2 because of the deletion of the “ry” from the source string. And the distance is 0 for the strings “check” and “check” because no edit is required.

The edit distance algorithm is stated as follows [20]:

1. Set n and m length of the source and target strings.
2. If $n = 0$, return m and exit.
3. If $m = 0$, return n and exit.
4. Construct a matrix containing $0..m$ rows and $0..n$ columns.
5. Initialize the first row to $0..n$.
6. Initialize the first column to $0..m$.
7. Examine each character of source string (i from 1 to n).
8. Examine each character of source string (j from 1 to m).
9. If $\text{source}[i]$ equals to $\text{target}[j]$, the cost is 0. If not, the cost is 1.

10. Set cell $d[i, j]$ of the matrix equal to the minimum of $d[i - 1, j] + 1$ (deletion), $d[i, j - 1] + 1$ (insertion) or $d[i - 1, j - 1] + \text{cost}$ (substitution).
11. After the iteration is completed, the distance is found in the cell $d[m, n]$.

According to the edit distance algorithm, the accuracy rate of an OCR system is:

$$R = \frac{m - d[m, n]}{m}, \quad (2.5)$$

where R is the accuracy rate, m and n are the length of the original and recognized text, and $d[m, n]$ is the distance of the two texts.

2.6 A Quick Review of NetLogo

NetLogo is an environment to develop agent-based model simulations. Uri Wilensky authored it in 1999 which it was designed in the spirit of the Logo programming-language first developed by Papert. NetLogo has a simple syntax and well-thought out system to create complex models. It consists of a world and four types of agents which are turtles, patches, links and the observer.

The world is the environment of the turtles, patches and links, and it can be either 2D or 3D. Turtles are the autonomous agents moving around the world. Patches are the squares composing the ground of the world. Turtles and patches have coordinate and color attributes. Communication between turtles are provided over links. Links do not have coordinates. The observer is able to control the world and the attributes of the agents.

A multi-agent modeling structure, the ability of embedding NetLogo in web pages and scripts, and open source extensions of this language facilitate solving complex problems and enable the observation of the application in different platforms.

In this study, the NetLogo platform is selected for the character recognition task since it provides the required environment to complete the project, which is a controllable agent (turtle) that can walk over the characters and a ground (world) to display the image.

2.7 Summary

In this chapter, we have overviewed the literature relevant for this project. It shows that human functions such as character recognition can be simulated by computers. Although there has been many studies for character recognition task, OCR systems can not reach a hundred percent accuracy

rate and still have problems on reading handwritten texts and recognizing similar characters such as “l” and “1” or “g” and “q”. OCR systems generally consist of four main steps and various algorithms are implemented in this steps.

Performance of the OCR systems can be evaluated using their accuracy rates. The Levenshtein method is used to find the accuracy rate of the OCR system by matching the recognized text with the original text.

This project is fundamentally based on a new approach for character recognition to reduce complexity of the task and to solve the similarity problem of characters. The approach proposed in this project uses the NetLogo environment to walk on the characters by controlling the agents.

Chapter 3

System Design

This chapter discusses the design structure of the OCR system developed in this project. The first section explains the required platforms to complete the project and their relationship. The second section describes the reasons for choosing NetLogo and the possible advantages of this language for OCR applications.

3.1 Structure of The System

The OCR system developed in this project uses three main platforms, the NetLogo and Java programming languages, and the MySQL server, to implement the agent based approach. According to our new agent based approach, recognizing characters by walking over them is the essential part of the approach and it is implemented in NetLogo. The other platforms are used to support NetLogo. Figure 3.1 shows the basic design of the proposed OCR system. According to Figure 3.1, the major steps of the OCR system are:

1. The user loads the image of the text document to the system using the interface of the Java. Also, the user loads the original text document to the system for evaluation of the results.
2. The user selects some image processing algorithms among several options and then starts the image processing based on the choices in Java.
3. After the image processing in Java is completed, Java writes the pixel information (color and location) of the processed image to the “common.dat” document and runs NetLogo.
4. NetLogo reads the information of the image from the “common.txt” document and applies the

agent based approach to the image. NetLogo determines each character according to the results of agent based approach from the MySQL database.

5. After the recognition process is completed, NetLogo writes whole recognized text to the “results.txt”.
6. Finally, Java reads the recognized text from the “results.txt”. Some postprocessing algorithms are applied to the text and the accuracy rate is found by comparing the recognized text with the original text loaded by the user.

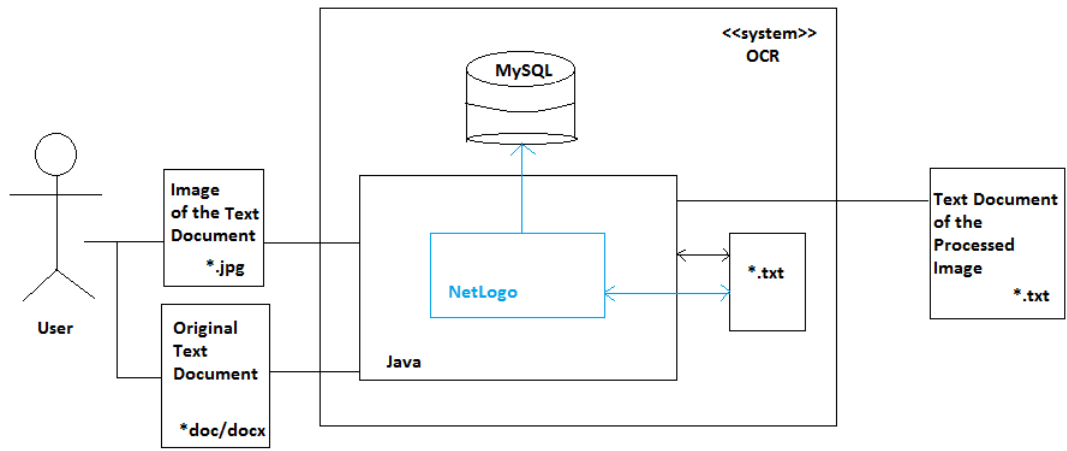


Figure 3.1: A basic structure of the system.

After an agent completes movement over a character, we have some results which give information about the character. The results of the movement are matched against features of the all characters in order to decide which character they belong. So a list is required that contains features of all characters (such as the edge count of the characters) to use in matching the movement results of NetLogo. Another platform is used to hold this list of the characters. This list can be stored as a procedure in NetLogo and therefore the requirement of another platform to hold the list could be removed. However the list consists of 74 characters and each character has more than one feature such as start point, finish point or edge count. Implementing this list inside NetLogo rises the complexity of the NetLogo code, requires long time to find a character and to update the list. Thus MySQL server is implemented in the system to contain the list. The reasons to choose MySQL are:

- NetLogo is able to connect to the server using a specific extension of NetLogo.

- It is an open source system and very fast at connecting to the database and responding to the queries.
- Updating the database tables is very easy.

NetLogo programs are called as models, and models have interactive interfaces. But NetLogo has limited interface components, and the implementation of some of the methods needed for image processing in an OCR system is difficult in NetLogo. This limitation gives rise to the necessity of a second programming language. The second language should be able to control NetLogo, should have various interface components as well as large libraries to implement OCR algorithms easily.

NetLogo can be run by another program running also on the Java Virtual Machine. Java is selected as the second programming language to invoke NetLogo, create the interface of the program with the help of functional components. Java runs NetLogo as embedded that allows the interface of the NetLogo model to be shown. Java can control NetLogo commands but NetLogo is unable to manage Java commands or to directly send information to Java. Both programming languages can read/write from texts. So when they need to share information, they can communicate through texts.

To sum up, the system divides the character recognition task into three main phases which are image processing, recognition and evaluation. The image processing phase is conducted by Java which prepares the image ready for recognition. The recognition phase is performed by NetLogo with the help of MySQL and this phase extracts the characters from the image. The evaluation phase analyzes the results of the proposed OCR system. To this aim, firstly the recognized text is improved by postprocessing algorithms and then the accuracy of the recognized text is calculated. This phase is also conducted in Java.

These phases and their methodologies are explained in later chapters.

3.2 NetLogo for OCR

NetLogo presents a simple environment that is composed of a world and turtles. The world is a two dimensional ground. The ground of the world consists of small, equally-sized squares which are called patches. Each patch has a specific color and coordinates. Turtles are agents in the world that can move over the patches.

This project implements OCR using the properties of NetLogo. To achieve this, the following correspondences are established:

- The image corresponds to the world of the NetLogo. Both the image and the world are two

dimensional and their sizes are flexible.

- The pixels of an image are the patches of the world. Both the pixels and the patches comprise a ground. Also both of them have specific color and coordinates.
- Feature extraction and recognition processes of the OCR system correspond to movement (walking) of a turtle over patches. So feature extraction and recognition occur simultaneously in NetLogo.

3.2.1 The Advantages of NetLogo

NetLogo environment provides some advantages in character recognition. Some of them are:

Functionality

- There is a rich set of commands to control the turtles using coordinates and color of the patches, and coordinates of the turtles. Thanks to these commands, getting information about the patches is faster and easier in NetLogo as an agent based programming language rather than the object oriented programming languages such as Java, as shown in Figure 3.2.

Java	NetLogo
<pre>// ** Origin Pixel ** p = pixel[i][j]; // ** Neighbors of The Origin Pixel ** p1 = pixel[i-1][j]; p2 = pixel[i-1][j+1]; p3 = pixel[i][j+1]; p4 = pixel[i+1][j+1]; p5 = pixel[i+1][j]; p6 = pixel[i+1][j-1]; p7 = pixel[i][j-1]; p8 = pixel[i-1][j-1]; // ** Black Neighbor Count of The Origin Pixel ** neighbor = p1 + p2 + p3 + p4 + p5 + p6 + p7 + p8;</pre>	<pre>;; Black neighbor count of the patch at (i, j) ask patches with [pxcor = i and pycor = j] [show count (neighbors with [pcolor = black])]</pre>

Figure 3.2: The codes to find counts of neighboring pixels that are black in Java and NetLogo.

- Movement (walking) results of an agent over a character such as patch count, start position or movement directions are specific to each character and so misrecognition between different characters are reduced in NetLogo.
- Spurious parts of characters affect the recognition process and they are common problems of OCR systems, particularly in recognition of the handwritten documents. Detecting spurious parts

between separate characters is possible in NetLogo thanks to agent based OCR approach as shown in Figure 3.3.

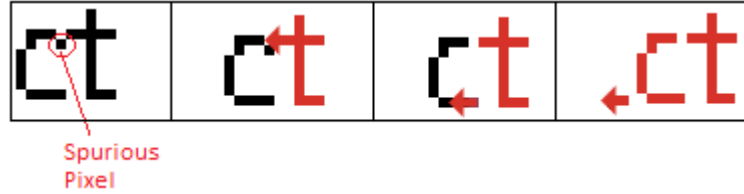


Figure 3.3: An example of cleaning spurious part of a character.

Our NetLogo implementation starts to read characters from the right bottom of the world. In Figure 3.3, the turtle first moved to the character “t”. Since NetLogo extracts features and recognizes the letter simultaneously, when the turtle finished moving over the letter “t”, it recognized the character as “t” letter. After that it accepts the first pixel as spurious and cleaned the spurious pixel between “t” and “c”, before starting to move over “c”.

Complexity

- As shown in Figure 3.3, NetLogo merges feature extraction, classification and recognition steps of OCR into one step. This reduces the complexity and saves time.
- Beside all advantages of NetLogo for OCR process, one of the most important advantages of NetLogo is allowing the observation of what the system is recognized step by step. Thus failure of the program is easily detected and new algorithms for recognition can be developed. We can clearly observe results of our algorithms.

Time

- NetLogo supports multi agent modeling that can run more than one turtle at same time. Thanks to this attribute, each turtle can be charged to read a line of the text. This can save substantial time for large size text documents.
- NetLogo is run as embedded from Java to display the interface of the NetLogo model in this project. If we do not want to display the interface, and only want to see the results, it is possible to run NetLogo in the background which is called in headless mode. This mode saves time and does not affect the interface of the main Java program.

3.3 Summary

In this chapter, we have explained the design of the OCR system and the advantages of using NetLogo for an OCR application. OCR is comprised of three main steps which are image processing, recognition and evaluation. The image processing and evaluation steps are conducted by Java and the recognition step is applied by NetLogo.

In the image processing step, some algorithms are applied to the image to make it suitable for recognition. The recognition step is to read characters from the image by matching movement results of the turtle against a character from the MySQL database. NetLogo can connect to the MySQL database and MySQL is very fast and easy for queries. And in the evaluation step, some postprocessing algorithms are applied to the recognized text and the accuracy rate of the text is calculated.

NetLogo has several advantages in perform agent based character recognition. It reduces the complexity of the OCR process, makes it easier to find solutions to solve similarity problems of characters and can save time thanks to multi agent capabilities.

Chapter 4

Image Preprocessing

In this chapter, the algorithms implemented in Java for the image processing step of the OCR system are introduced. The first section explains the structure of the image processing in the OCR program. The second section introduces the convolution filters and implementation of them in Java. The third and fourth sections discuss the thinning algorithms and the thresholding algorithms. The last section presents a new approach for the thinning algorithms.

4.1 Image Processing

The aim of the image processing is to prepare the image ready for processing and recognition in NetLogo. This step is mostly required for the documents that are handwritten, damaged, blurred or noisy. For the typewritten documents, if the paper quality is high, some algorithms of this step are mostly not needed. So, in the program, some preprocessing algorithms are optional. The user can decide which image processing algorithms are required according to the paper quality.

The methods used in the image processing are Convolution Filters, Thresholding and Thinning Algorithms. Only some of the convolution filters are optional in the program. Thresholding and Thinning Algorithms are always applied to the image. The approach in NetLogo requires the image in only black and white color, which is provided by thresholding algorithms, and the lines of the characters in the image to be one pixel width, which is provided by thinning algorithms.

In Figure 4.1, the optional image processing methods are shown which are located in the first screen of the program.



Figure 4.1: The first screen of the interface of the Java program.

According to Figure 4.1, the user loads the image of the text document into the system using the “Load Image” button and the loaded image is displayed on the left of the buttons. Also, the original text document is loaded using the “Load Text” button. Then if the user selects the “Image Processing” button, the options for image processing are displayed as shown in Figure 4.2 which are convolution filters (noise removing and edge detection), thinning algorithms and thresholding methods. These are the methods implemented by the system:

- Noise Removal: Mean, Median and Gaussian filters.
- Edge Detection: Laplacian, Prewitt, Roberts and Sobel filters.
- Thinning: Zhang-Suen and Hilditch algorithms.
- Thresholding: Otsu and Local methods.

The user can pick one of the method from each category such as Mean, Sobel, Hilditch and Otsu methods or Median, Prewitt, Zhang-Suen and Local methods. Thus when the user clicks the “Resolution” button, the selected methods are applied to the image and then the NetLogo model is displayed on the Java interface. However, if the user does not select the “Image Processing” button, only Otsu thresholding and a new thinning algorithm developed in this project are applied to the image.

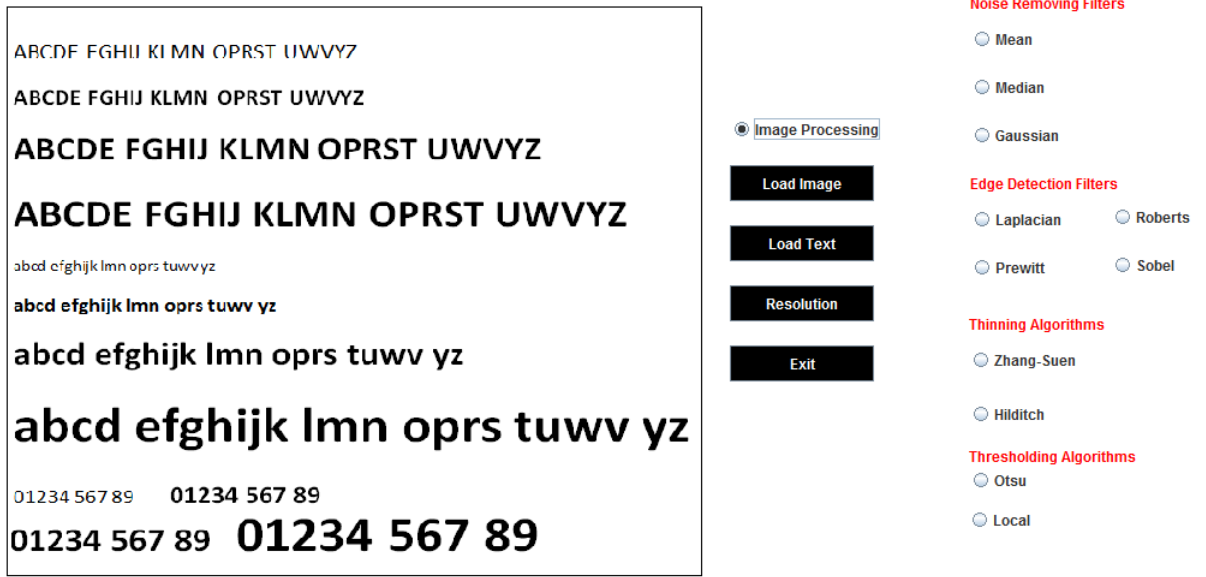


Figure 4.2: The optional image processing methods.

4.2 Convolution Filters

The convolution operation is implemented using ConvolveOp class of Java [4]. A kernel, which is a 2D spatial operation, is used for the convolution process. The kernel convolves a pixel by multiplying the value of the given pixel by the values of the surrounding (neighbor) pixels. Kernels of different size and values create various effects on images. The output image, which is the result of convolution operation, is called a filtered image.

To apply the convolution operation, images must be buffered. The buffered image is updated after each convolution operation.

4.2.1 Sharpening Filter

This filter exaggerates features of the original image. It is used to enhance the edges in the image. This filter is not optional in the program because it facilitates the thinning operation and creates more efficient results as shown in Figure 4.3.

A 3×3 sharpening kernel is selected in the program as shown in Figure4.4.

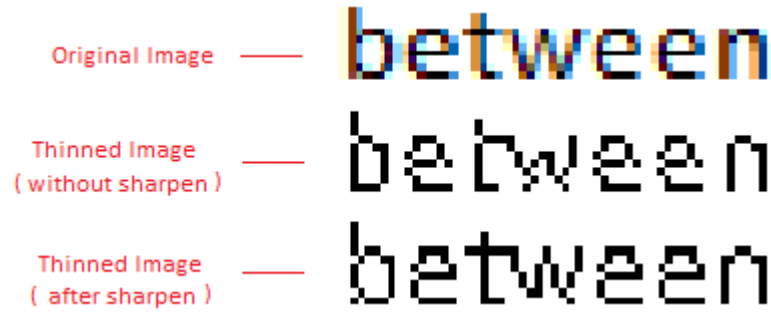


Figure 4.3: An example of the effect of a sharpening filter.

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Figure 4.4: The kernel of the sharpening filter.

4.2.2 Edge Detection Filter

The edge detection filters are optional in the program. While they determine the edge pixels in the image, they clean the other pixels from the image. The cleaned pixels may, however, be important details required for thinning algorithms.

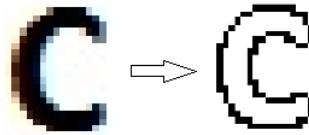


Figure 4.5: An example result of the edge detection algorithm on a bold character.

This problem may arise in bold letters, where the edge detection algorithm detects a line as two lines because of the width of the character, as shown in Figure 4.5. Then the thinning algorithm may recognize these edges as two separate candidate characters. Four type of edge detection filters are implemented within the program. These are Laplacian, Prewitt, Roberts and Sobel filters [26]. The effects of these filters on the same image are presented in Figure 4.6.

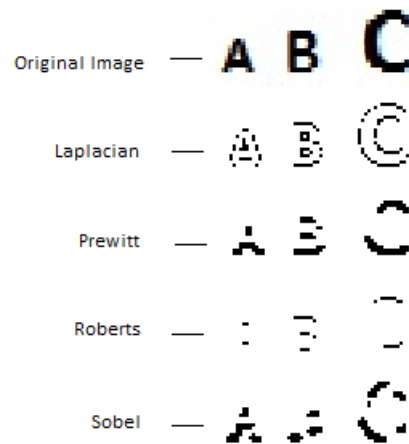


Figure 4.6: An example of four edge detection filters.

4.2.3 Noise Removing

Similarly to edge detection filters, noise removal filters are not compulsory in the preprocessing step. To remove noise from the image, this filter blurs the image. If the image does not contain too much noise, this filtering may lead to blurring the edges and removing important details from the image. This situation also depends on the type and size of the kernel. If the size is large, the blur on the image will be higher.

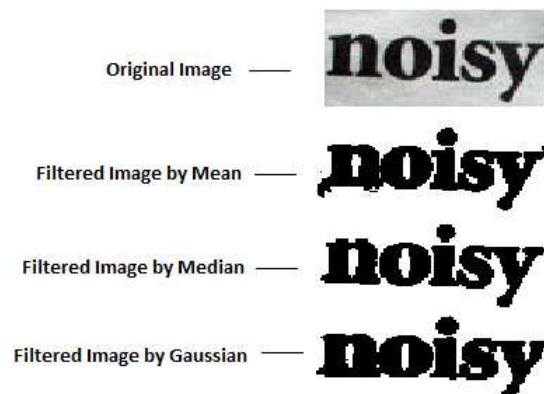


Figure 4.7: An example of three noise removal filters.

In the program, there are three options for noise removal filters. These are Mean, Median and Gaussian Smoothing filters [13]. Figure 4.7 displays a comparison of these filters on the same image. If the image requires this filtering, Gaussian smoothing kernel can be a better choice than the other

filters because it preserves the edges on the image.

4.3 Thresholding Algorithms

Thresholding is a non-linear operation to convert a gray scale image into a binary image. Two types of thresholding methods, global and local (adaptive), are adopted in the program. The Otsu thresholding method is picked for global thresholding [17]. The steps of this algorithm are:

1. Divide the pixels into two clusters (foreground and background) according to the threshold value.
2. Calculate the mean of each cluster.
3. Find the difference between the means and square the difference.
4. Multiply by the number of pixels in one cluster times the number in the other.

The Otsu algorithm is fast and easy to implement in Java. It is mostly suitable for typewritten and good quality papers because there is not a substantial unbalance between foreground and background pixels. The thresholding value of Otsu is generally suitable for every part of the image.

However if the document is handwritten or low paper quality, the Otsu method may fail due to the high unbalance between foreground and background pixels. So if the user does not choose a preprocessing step, the Otsu method is adapted as a compulsory thresholding method. If the user selects a preprocessing step, there is a second option for thresholding which is the Local thresholding method [16].

The Local method finds a thresholding value for each pixel instead of a constant one like in the Otsu method. The localized thresholding deals with the neighborhood of a given pixel. The size of the neighborhood, n , depends on the size of area of interest in the image. n should be small enough to not take into account unrelated pixels and to find the optimum thresholding value.

However, if n is selected too small, the algorithm may be too sensitive to noise and lead to over-segmentation. Likewise, if n is chosen very large, then the method works as a global thresholding algorithm. The n value is set to 8 in the program. The steps of this algorithm are:

1. Calculate intensity of each neighbor pixel of the given pixel and find the sum of these intensities.
2. Divide the sum value by the neighbor count (which is 8). The result is the threshold value (T) of the given pixel.

3. If the intensity of the given pixel is higher than T , set the value for this pixel to 1 (foreground).
If not, set the value to 0 (background).
4. Repeat these steps for each pixel in the image.

The result values of all pixels (1 or 0) are recorded to a two dimensional array. This array is used in thinning algorithms and for the transferring of the image into a NetLogo world.

4.4 Thinning Algorithms

Some of the famous thinning algorithms, Zhang-Suen and Hilditch [51], are implemented in the program. They are parallel and iterative thinning algorithms and their structures and rules are similar. They efficiently work on large sized, bold characters. However, they can clear the connectivity between parts of a character or a whole part of a character in documents that are written in small size.

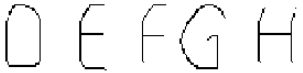
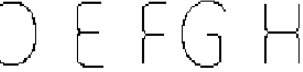

Original Image —	D E F G H	NetLogo Simulation
Zhang-Suen —		NetLogo Simulation
Hilditch —		NetLogo Simulation
The new method —		NetLogo Simulation

Figure 4.8: An example of comparison of thinning algorithms on bold characters.

Original Image —	Movement:	be turn exactly	100
Zhang-Suen —	Movement	be turn e, actly	100
Hilditch —	Movement	be turn e, actly	100
The new method —	Movement:	be turn exactly	100

Figure 4.9: An example of comparison of thinning algorithms on different types of characters.

For the program developed within this project to work, all parts of a character should be connected. This is required because a character will be recognized using transition information between its parts. If there is a gap between two parts of a character, the program will accept the character is completed before the gap and a new character will be sought after the gap.

Thinning algorithms mostly focus on cleaning spurious parts but they may also clean required details too. Because the approaches of these algorithms are not suitable to the approach adopted in this project, a new thinning algorithm is developed to fulfill the requirements. While the new thinning algorithm mostly saves connectivity of a character, the spurious parts may remain. Unlike previous thinning algorithms, this algorithm can work better on small sized texts. But it is not as good with bold, large sized characters. Comparisons of Zhang-Suen, Hilditch and the new algorithm on different type characters are shown in Figure 4.8 and Figure 4.9.

4.5 A New Approach for Thinning

The aim of this algorithm is detecting connectivity of the neighborhood of a given pixel. A pixel has 8 neighbor pixels that are directly connected to it. If all black neighbor pixels are connected to each other, we called them the “encasement” of the given pixel.

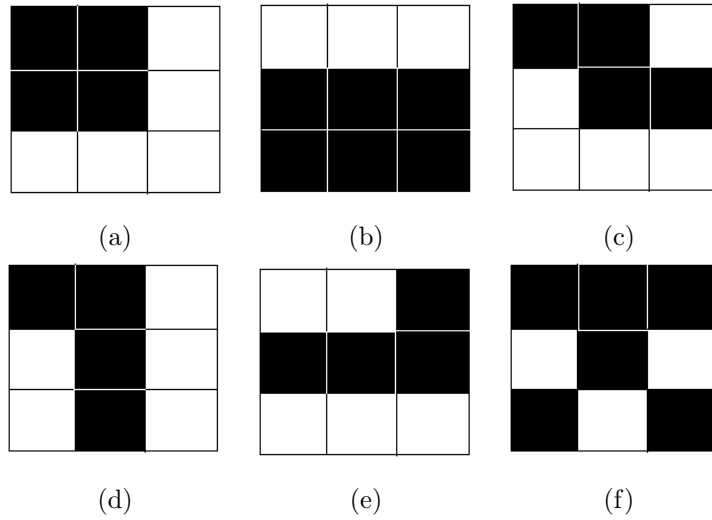


Figure 4.10: An example of two convolution filters.

In Figure 4.10, the center pixel has an encasement in the (a), (b) and (c) images because each neighboring pixel of the center pixel are connected to at least 1 other black neighboring pixel either

vertically or diagonally. But, in the (d), (e) and (f) images, there is not an encasement of the center pixel.

The main rule of this algorithm is that if a pixel has an encasement, then clean the given pixel. This algorithm works sequentially and iteratively. Namely, when it decides on a pixel to delete, the algorithm cleans it at same time. This process is applied to all pixels in the image. When all pixels are processed, if any pixel is deleted, the algorithm is restarted. This iteration continues until no pixel is deleted.

4.5.1 Methodology of The Algorithm

The new algorithm uses the 2D integer array created by thresholding algorithms. Each item of the array corresponds to a pixel of the image. So row and column information of an item in the array is equal to the coordinates of a pixel in the image. And the value of the item is either 1 (black pixel) or 0 (white pixel). If the algorithm decides on a pixel to delete, it just changes the value of corresponding item in the array from 1 to 0. After the thinning process is completed, the 2D integer array is used by NetLogo for the recognition phase.

The new algorithm enumerates all neighbors of the given black pixel from $P1$ to $P8$ and finds the count of black neighbors of it. Then the first black neighbor is chosen as the start pixel based on the ordered pixel sequence. The start pixel is used to search the connectivity between the neighbors.

The Connectivity Rule

The black neighbors count of a given pixel is n and each black neighbor of a given pixel is compared with the next black neighbor according to the pixel sequence. If both the differences in row values and differences in column values of the two pixels are smaller than 2 and greater than -2 , there is connectivity between the two pixels. This operation is repeated $n - 1$ times and so applied to all black neighbors. If all of them satisfy this rule, there is connectivity between neighbors and the given pixel has the encasement.

For example, assume an image template as shown in Figure 4.11, where the given pixel is P and the neighbors are enumerated from $P1$ to $P8$. The 2D array of this image is as shown in Figure 4.11, where the black neighbors are ($P2$, $P3$ and $P4$) and the given pixel is 1 and the remaining white pixels are 0. The row and column values of each item of the array are shown with black and red values in Figure 4.11. According to the algorithm:

- The start pixel is $P2$ and n is three.

- $P2_{row} - P3_{row} = 0$ and $P2_{column} - P3_{column} = 1$. Both results are smaller than 2 and greater than -2 , so $P2$ and $P3$ pixels are connected.
- $P3_{row} - P4_{row} = -1$ and $P3_{column} - P4_{column} = 0$. Both results are in the required range, so $P3$ and $P4$ pixels are connected.

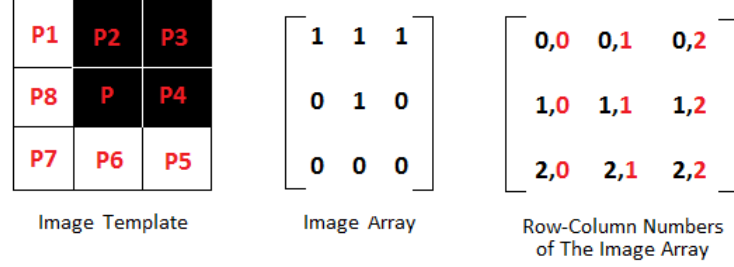


Figure 4.11: An example of an image template and its array information.

Since each black neighbor is connected to the next one according to their numeration, P has the encasement and it is deleted for thinning. If instead of $P4$, the $P5$ pixel was full (black), the connectivity rule was not satisfied, because the row difference between $P3$ and $P5$ is -2 .

In this algorithm, the start pixel is important to determine the connectivity. For example, in Figure 4.12, although $P2$, $P3$ and $P8$ pixels are connected to each other, in the algorithm, the result is not the same due to the enumeration of pixels. According to the algorithm, the start pixel is $P2$, so the $P2$ and $P3$ pixels and then the $P3$ and $P8$ pixels are compared for connectivity. Because the row difference between $P3$ and $P8$ is -2 , they are accepted as unconnected. But if we chose start point $P8$ instead of $P2$, the rules would be satisfied and the result would be connected.

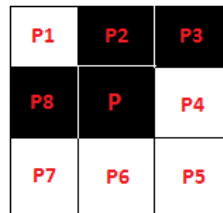


Figure 4.12: An example of an image template.

To solve this problem and to modify the algorithm to all types of image templates, four different enumeration types are developed according to location of the first pixel ($P1$) and enumeration direction as shown in Figure 4.13.

In Figure 4.13, the start pixels are at the top in Type I and Type II. But in Type I, pixels are enumerated in a clockwise manner and in Type II, enumeration is in the reverse direction. In Type III and Type IV, the start pixels are at bottom, but in Type III the direction is in a clockwise manner while in Type IV, enumeration is in the reverse direction. A different enumeration type is chosen according to the neighbors count of a given pixel.

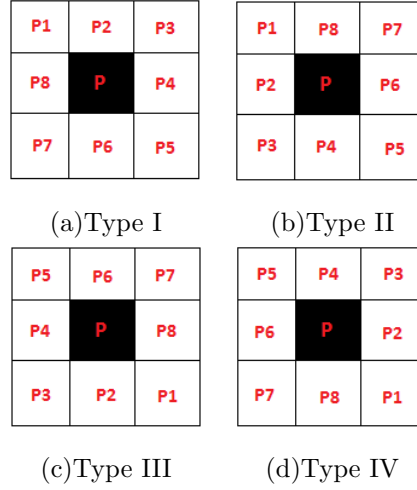


Figure 4.13: The types of enumeration of neighbor pixels.

Also, only for the pixel whose neighbors count is 2, the adjacency rule as follows is applied:

- Enumerate neighbors using one of the enumeration type.
- If the numeration of the two neighbors are consecutive (such as $P1$ and $P2$ or $P5$ and $P6$), the two pixels are connected and the given pixel has the encasement.

In Figure 4.14, while the image (a) has the encasement, there is not an encasement in the image (b) due to the non-consecutive neighbors.

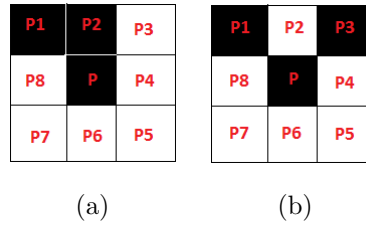


Figure 4.14: An example of pixels with two neighbors.

Table 4.1 shows the enumeration types and rules according to the neighbor counts.

Black Neigh. Count	Enumeration Type	The Rule
0	-	-
1	-	-
2	All types	The adjacency rule
3	Type I	The connectivity rule
4	Type I	The connectivity rule
5	All types	The connectivity rule
6	All types	The connectivity rule
7	All types	The connectivity rule
8	-	-

Table 4.1: The rules of the new thinning algorithm.

To sum up, the new thinning algorithm can be stated as follows:

1. Find a pixel whose color is black.
2. Calculate black neighbor counts (n) of the pixel.
3. If $n = 2$, apply one of the enumeration types and then use the adjacency rule. If the rule is satisfied, delete the pixel.
4. If $2 < n < 5$, apply enumeration Type I and then use the connectivity rule. If the rule is satisfied, delete the pixel.
5. If $5 \leq n < 8$, apply all enumeration types until one of them satisfy the connectivity rule. If one of them satisfies the rule, do not apply the remaining enumeration types and delete the pixel. If none of them satisfy the rule, do not delete the pixel.
6. Repeat steps 1 to 4 until all pixels of the image are processed.
7. If at least one of the pixels is deleted, repeat from step 1.

Analysis of The New Thinning Algorithm

The main purpose of this algorithm is saving connectivity between parts (edges) of a character. It fulfills the expectancy of this project and thins the image as required for the subsequent NetLogo algorithms. Especially, it works efficiently on small sized characters and preserves the important

details of characters which will be required for recognition in NetLogo. However, it is not so good at thinning large size characters.

Implementation of this algorithm is more complex than the Zhang-Suen and Hilditch thinning algorithms. While Zhang-Suen or Hilditch is composed of less than 150 lines code in Java, the new algorithm is composed of more than 300 lines of code. Although implementation of the new algorithm is more complex, there is not a significant running time difference between the new algorithm and the others.

4.6 Summary

In this chapter, we have introduced the algorithms used for image processing. Edge detection and noise removal filters are implemented into the system as optional. However, sharpening filter, thresholding and thinning algorithms are compulsory in the image processing step.

The thresholding algorithms convert the image into binary format, and Local and Otsu thresholding methods are implemented into the system. The Otsu method creates better results than the global method in good quality papers.

The thinning algorithms reduces the width of the lines to one pixel by deleting foreground pixels of the image. The Zhang-Suen and Hilditch algorithms are adopted to the system. They are good at thinning large sized or bold characters. But they can clean important details of small sized characters. Thus a new thinning algorithm is developed in order to protect connectivity between edges of a character.

The new algorithm searches the pixels who have an “encasement” and delete them. To detect the encasement, it enumerates neighbors of the given pixel and uses the connectivity and adjacency rules according to the black neighbor count of the pixel. There are four type enumerations. Two of them enumerate neighbor pixels in a clockwise manner and the other two types enumerate in the reverse direction.

If all black neighbors of the given pixel are connected to each other, the pixel has the encasement and satisfies the connectivity rule. If numbers of the black neighbors are consecutive, the pixel has the encasement and satisfies the adjacency rule. The new thinning algorithm works efficiently on small sized characters but not as good with large characters.

Chapter 5

Agent-based OCR in NetLogo

This chapter introduces the NetLogo model that is developed for character recognition in this project. The first section explains transfer of the processed image from Java to the NetLogo environment. In the second section, the new agent based approach for OCR and the structure of the NetLogo model are presented.

5.1 Displaying the Processed Image in the NetLogo World

As mentioned in section 4.5.1, the 2D integer array, which holds the color value of each pixel as either 1(black) or 0(white), is used for transferring the image into NetLogo. When Java completes the thinning process, it writes the final array line by line to the “common.dat”. Also, Java writes the dimensions of the image (width and height) to the “dimension.txt” and then runs the NetLogo model.

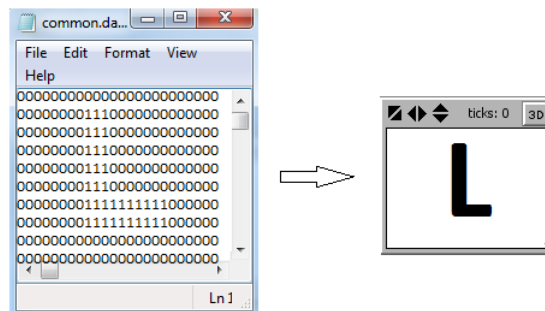


Figure 5.1: An example of transferring the image into the NetLogo world.

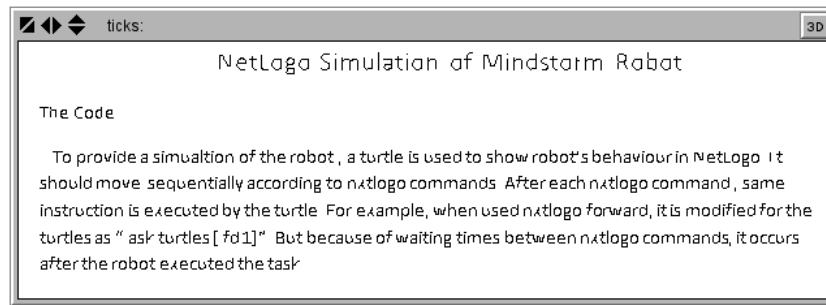
Firstly, the NetLogo model reads the “dimension.txt” and creates the world (consisting of patch) agents in NetLogo according to the dimensions. Then, the model reads the “common.dat” line by line. Each line of the document corresponds to each row of the world and each item of a line corresponds to each patch of the row. So if the value of an item is 1, the corresponding patch in the world is colored as black. If the value is 0, the patch is colored as white as shown in Figure 5.1. Thus, when the model read all lines of the document and created the patches based on the lines, the whole text would be displayed in the world of NetLogo as shown in Figure 5.2.

NetLogo Simulation of Mindstorm Robot

The Code

To provide a simulation of the robot, a turtle is used to show robot's behaviour in NetLogo. It should move sequentially according to nxtlogo commands. After each nxtlogo command, same instruction is executed by the turtle. For example, when used nxtlogo:forward, it is modified for the turtles as “ask turtles [fd 1]”. But because of waiting times between nxtlogo commands, it occurs after the robot executed the task.

(a) Original image.



(b) Displaying the thinned image in NetLogo world.

Figure 5.2: An example of transferred image to NetLogo world.

The patch structure of NetLogo facilitates conversion of the image to the NetLogo world. There is no fixed size limits of the world; it is possible to enlarge the world as required. But the memory (RAM) of the system may not be enough to create very large size worlds. In other words, the NetLogo model may exceed the memory limit of the system and may not allow the world to be created in the required size. In this project, the NetLogo world size can be raised up to the 2000×2000 dimension.

5.2 Methodology of the Agent-based Approach for OCR

The new approach conducts the OCR process in four main steps. These are:

1. line determination;
2. character recognition;
3. character matching from the database;
4. displaying results.

The turtle, which is used for the movement over the characters, is located at the right bottom of the world. The NetLogo model determines all lines of the text and the turtle is located on each line in turn from bottom to top. When the turtle is located on a line, it moves forward from right to left. While it moves on the line it extracts the characters and tries to match them from the database. And the results are simultaneously displayed on the NetLogo interface.

5.2.1 Line Determination

The turtle moves across each line of the text once from right to left starting from the right most margin. To this aim, before the turtle starts its movement, lines of the text are determined using a threshold algorithm in NetLogo. The thresholding algorithm uses a threshold value that can be changed by the user with the help of a threshold slider on the NetLogo interface. The most effective threshold value is 4 for the program. The steps of the thresholding algorithm are:

1. Set a threshold value using the threshold slider, which is T , and locate the turtle on the bottom row of the NetLogo world.
2. Record coordinates of the current row.
3. Count all black patches on the current row, which is n .
4. Record coordinates of the upper row and locate on this row.
5. Count all black patches on the current row, which is m .
6. If $\frac{m}{n} > T$, mark the current row as a line of the text.
7. If the current row is not the top row of the world, locate on the upper row and repeat from step 2.

The accuracy of the algorithm is very high and performs the task in a short time. For example, it finds all lines of a text, which consists of 30 lines, in 5 seconds. And the accuracy of the detected lines depend on the threshold value. If the threshold value is chosen higher than required, the algorithm may miss some lines.

5.2.2 Character Recognition

Thanks to the line determination method, the turtle is able to start to move over a character from the right bottom of the character. A method is developed in NetLogo for recognizing the characters. The basic algorithm of the method is stated as following:

1. Locate the turtle on the right of the bottom line of the world.
2. Move left one step on the line until a black patch is found ahead.
3. Mark the black patch as the start point of a character and locate the turtle on the start point.
4. Find a black neighbor of the located patch, move on the neighbor patch and change the color of it from black to red. According to the direction of the neighboring patch, set the heading in that direction.
5. Record the heading and increase visited patch count 1.
6. Repeat from step 4, until there is no black neighbor of the located patch.
7. Mark the located patch as the finish point of the character.
8. If the finish point is not at end of the line, repeat from step 2.
9. Move on the next upper line and repeat from step 2.

Methodology of the Character Recognition Algorithm

The frequently used terms in the algorithm are described as following and using these terms the pseudocode of the character recognition method is presented in Appendix A.

Start Point: While the turtle is moving over the selected row, the turtle marks the first black patch on the row as the start point of a character. And it changes the color of the start point from black to blue. Each character has a start point.

Finish Point: Likewise with the start point, the turtle records a finish point for each character. When it completes the movement over a character, it relocates at the finish point of the character and continues to move on the row to find the next start point.

Edge: An edge is composed of connected black patches which are on the same X coordinate or Y coordinate. For example, in Figure 5.3, character “i” has one edge, since the connected patches are on the same X coordinate. And character “n” has three edges: the patches are on the same X coordinate for number 1 and 3 edges, and the patches are on the same Y coordinate in number 2 edge.

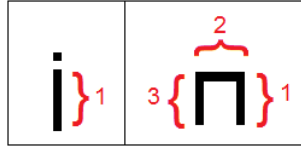


Figure 5.3: An example of edges.

Also if the patches are not on the same X or Y coordinate, but they are connected to each other via a diagonal, these patches also consist of an edge as shown in Figure 5.4.

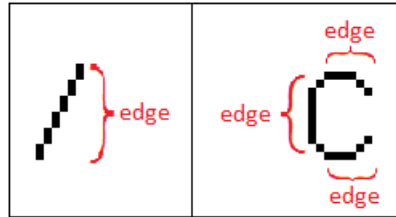


Figure 5.4: An example of edges whose patches are connected as diagonal.

Corner: The turtle can move in four directions which are top, bottom, right and left (these correspond to turtle headings of 0° , 180° , 90° and 270° respectively). Until the turtle reaches a corner, it moves in a single direction. When it comes to a corner, it changes its direction according to the position of the corner. A corner consists of three edges and satisfies one of the conditions below.

Condition 1: If any two of the three patches are on the same X coordinate and any two of the three patches are on the same Y coordinate, there is a corner as shown in Figure 5.5.

In Figure 5.5, patch 1 and 2 are on the same X coordinate and patch 2 and 3 are on the same Y coordinate. So when the turtle is moving from bottom to top, it recognizes there is a corner when it comes to patch 3. And it changes its direction from bottom to right.

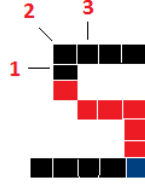


Figure 5.5: An example of corners which are based on Condition 1.

Condition 2: If any two of the three patches are on the same X or Y coordinate and the third patch is connected to them via a diagonal, there is a corner as shown in Figure 5.6.

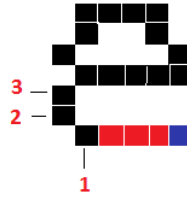


Figure 5.6: An example of corners which are based on Condition 2.

In Figure 5.6, patch 2 and 3 are on the same X coordinate while patch 1 is connected to patch 2 via a diagonal. So when the turtle is coming from right to left direction, it recognizes there is a corner when it comes to patch 3. And it changes its direction from left to top.

Branch: Branches are patches of a character which present more than one option for the next step of the turtle. The color of the branch patches is left as black. A patch can be a branch point if one of the conditions in below are satisfied.

Condition 1: Assume the turtle's current direction is top. In the next step of the turtle, if there is a black patch at the same direction with the turtle, which is top, and there is at least one corner, the current patch where the turtle moves to next is a branch point as shown in Figure 5.7.

Condition 2: If there is no black patch at the direction of the turtle, but there are at least two corners, the current patch where the turtle moves to next is a branch point as shown in Figure 5.8.

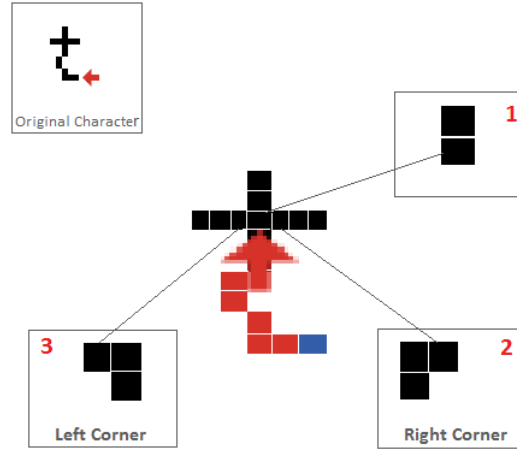


Figure 5.7: An example of a branch point which satisfies Condition 1.

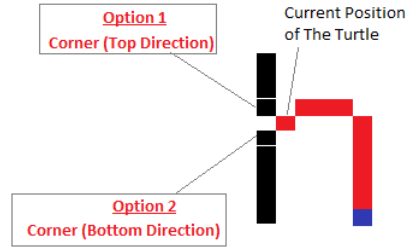


Figure 5.8: An example of a branch point which satisfies Condition 2.

Turtle Movement String: The turtle movement string is used to hold the change of direction names in the order that the turtle moves over a character. Each character has its own turtle movement string and it is used to recognize the character. In the turtle movement string, abbreviations of directions are used. These are :

- “l” : The turtle heading is 270° (moving westwards).
- “t” : The turtle heading is 0° (moving northwards).
- “r” : The turtle heading is 90° (moving eastwards).
- “b” : The turtle heading is 180° (moving southwards).
- “*” : The turtle is on a branch point.

Figure 5.9 presents movement of the turtle over some characters step by step and shows the change in edge count, turtle movement string and color of the patches in this period.

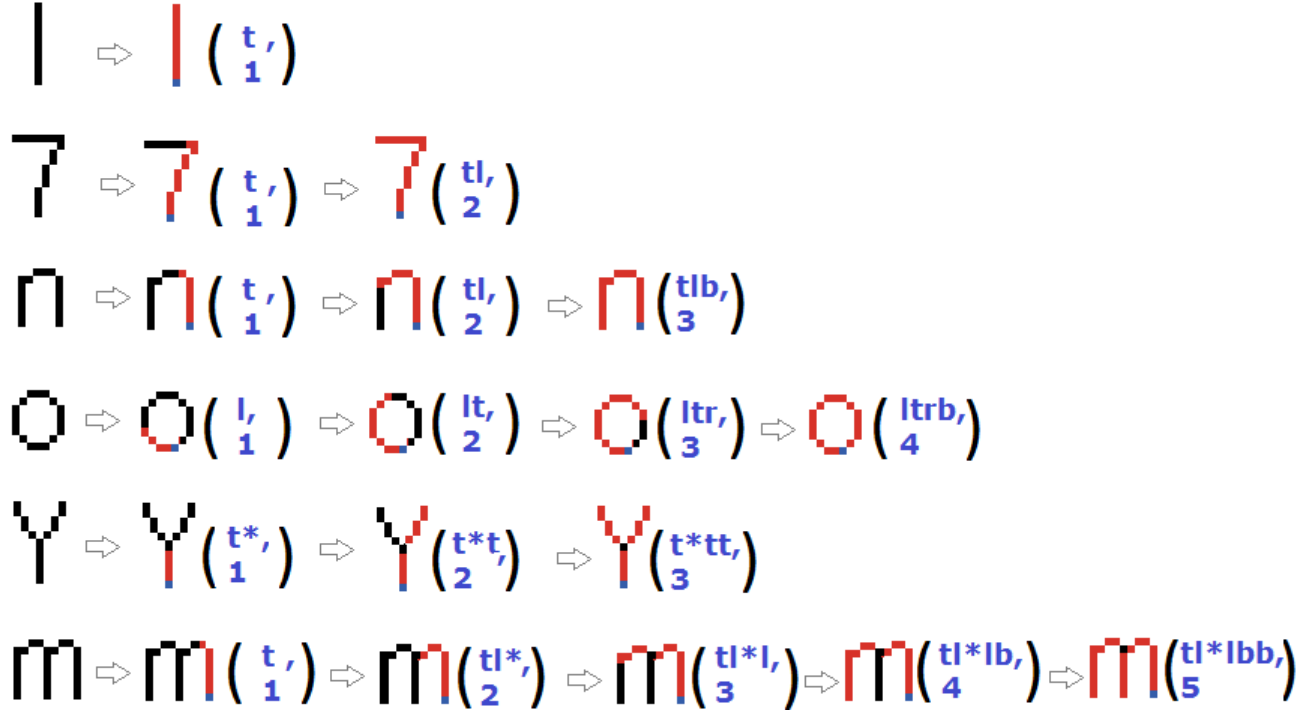


Figure 5.9: Example turtle movement strings and edge counts of some characters.

According to Figure 5.9, for character “I”, the turtle moved northwards and completed the character. Thus the turtle movement string of the character was only “t” and the edge count was 1. For character “n”, the turtle moved northwards, then westwards and finally southwards. Thus the turtle movement string of the character was “tlb” and the edge count was 3. On character “Y”, the turtle first moved northwards and found a branch point which consists of two options. So the turtle movement string became “t*”. Then the turtle moved to first option and the turtle movement string was updated as “t*t”. After that the turtle moved to second option and completed the character. Finally, the turtle movement string of the character was “t*tt” and the edge count was 3.

Letter Height: The letter height is the length of a character.

Line Height: The line height is the length of a line. If the letter height of a character is smaller than the line height, the character is marked as a lower character. If not, the character is marked as an upper character.

5.2.3 Character Matching from the Database

Character recognition is provided by matching the results of the turtle movements over the character against those for a character from the database. To this aim, MySQL server is adopted for the system. A MySQL extension in NetLogo, which is developed by Ll [28], is used to directly connect to the database from the NetLogo model. In this project, two databases are created for specific fonts which are “Gulim” and “Bitstream Vera Sans Mono” as shown in Table 5.1. The reason to choose this particular font types is that there is no connection between the characters in this font types. In other words, if two characters touch each other, when the turtle starts its movement, it continues to move on the second character since they are connected. Thus the turtle accepts two characters as a single character. In the font types selected in this study, characters do not touch each other and so the turtle can separately move on each character.

Table 5.1: Database of Gulim and Bitstream Vera Fonts.

Id	Gulim					Vera				
	Name	Type	Edge	Movement	Siml.	Name	Type	Edge	Movement	Siml.
1	A	upper	5	t*tlbl	##	A	upper	5	t*tl*tl	
2	B	upper	6	lt*trbr		B	upper	6	lt*trbr	
3	C	upper	4	ltrt		C	upper	4	ltrt	
4	D	upper	4	ltrb	#	D	upper	4	ltrb	#
5	E	upper	5	lt*trr		E	upper	5	lt*trr	
6	F	upper	4	t*trr	#	F	upper	4	t*trr	#
7	G	upper	7	t*tlbltr	#	G	upper	4	ltrt	#
8	H	upper	5	t*tl*tb		H	upper	5	t*tl*bt	
9	I	upper	1	t		I	upper	3	ltr	#
10	J	upper	3	lrr	#	J	upper	1	t	#
11	K	upper	5	t*tb*bt		K	upper	5	t*tb*bt	
12	L	upper	2	lt		L	upper	2	lt	
13	M	upper	7	t*tblt*tb		M	upper	7	t*tb*btlb	
14	N	upper	5	t*tt*tb		N	upper	6	t*tltlb	
15	O	upper	4	ltrb		O	upper	4	ltrb	
16	P	upper	5	t*trbl		P	upper	5	t*trbl	
17	Q	upper	6	t*tlbrt		Q	upper	5	ltrbb	

Continued on next page

Table 5.1 – continued from previous page

Id	Gulim					Vera				
	Name	Type	Edge	Movement	Siml.	Name	Type	Edge	Movement	Siml.
18	R	upper	5	t*tlbl	###	R	upper	5	t*tlbl	
19	S	upper	5	ltltr		S	upper	4	tltr	
20	T	upper	3	t*rl		T	upper	2	tr	
21	U	upper	3	ltt	#	U	upper	3	ltt	
22	V	upper	3	t*tt		V	upper	3	t*tt	
23	W	upper	7	t*tt*tb*bt		W	upper	8	t*tlb*bt	
24	X	upper	4	t*ttb		X	upper	5	t*tlbl	#
25	Y	upper	3	t*tt		Y	upper	4	t*tl	#
26	Z	upper	3	ltl		Z	upper	2	lt	
27	a	lower	7	t*tlbltr		a	lower	6	t*tltr	
28	b	lower	5	lt*rbt		b	lower	6	l*tltl	
29	c	lower	4	ltrt		c	lower	3	ltr	
30	d	lower	6	t*tbltr		d	lower	5	t*tltr	
31	e	lower	7	lt*trblt		e	lower	7	lt*trbrt	
32	f	lower	4	t*trr		f	lower	4	t*trr	
33	g	lower	6	lt*tlbr	#	g	lower	6	t*tlrb	#
34	h	lower	4	tl*tb		h	lower	4	tl*tb	
35	i	lower	1	tp		i	lower	2	ltp	
36	j	lower	1	tp	#	j	lower	2	tb	
37	k	lower	5	t*tl*tb		k	lower	6	lt*rb*bt	
38	l	lower	1	t		l	lower	2	lt	##
39	m	lower	5	tl*lbb		m	lower	5	tl*lbb	
40	n	lower	3	tlb		n	lower	4	tl*lb	
41	o	lower	4	ltrb		o	lower	4	ltrb	
42	p	lower	5	l*trbb		p	lower	7	l*bt*rbtr	
43	q	lower	5	t*tlbr		q	lower	3	t*tlrb	
44	r	lower	2	tr		r	lower	4	t*rt	
45	s	lower	5	ltltr		s	lower	4	tltr	
46	t	lower	4	lt*tr		t	lower	4	lt*tr	

Continued on next page

Table 5.1 – continued from previous page

Id	Gulim					Vera				
	Name	Type	Edge	Movement	Siml.	Name	Type	Edge	Movement	Siml.
47	u	lower	5	t*tblt	#	u	lower	3	t*tl	#
48	v	lower	3	t*tt		v	lower	3	t*tt	#
49	w	lower	7	t*tt*tb*bt		w	lower	8	t*tlb*bt	
50	x	lower	3	t*tb		x	lower	3	l*lt	
51	y	lower	4	lt*tt		y	lower	5	t*tlb	
52	z	lower	3	ltl	##	z	lower	2	lt	#
53	0	number	4	ltrb		0	number	4	ltrb	##
54	1	number	2	tl		1	number	3	ltl	#
55	2	number	4	ltlb		2	number	5	ltrtl	
56	3	number	5	lt*tll		3	number	6	tl*tlr	
57	4	number	6	t*tlbrr	#	4	number	6	t*tlbrr	
58	5	number	5	ltltr		5	number	6	lt*trl	
59	6	number	6	lt*rbtr		6	number	6	lt*rbtr	
60	7	number	2	tl		7	number	1	t	
61	8	number	6	lt*trrt	#	8	number	6	lt*trbr	#
62	9	number	6	lt*tlbr		9	number	6	t*tlbrb	#
63	.	punc	1	dot		.	punc	1	dot	
64	,	punc	1	coma		,	punc	1	coma	
65	;	punc	1	semicoma		;	punc	1	semicoma	
66	:	punc	1	colon	#	:	punc	1	colon	
67	/	punc	1	rightslash		/	punc	1	rightslash	
68	\	punc	1	leftslash		\	punc	1	leftslash	
69	(punc	2	leftpar		(punc	1	leftpar	
70)	punc	1	rightpar)	punc	1	rightpar	
71	[punc	4	ltr	#	[punc	4	trbr	#
72]	punc	5	t*tlbl]	punc	1	squarepar	
73	{	punc	3	t*tl		{	punc	4	trbr	
74	}	punc	3	t*tr		}	punc	4	t*trb	

In Table 5.1, the name column corresponds to the original character, the type column can be punctuation, number or letter(upper or lower), the edge column corresponds to edge count of the character and the movement column shows turtle movement string of the character. Some characters have the same turtle movement strings such as the turtle movement string “t*tt” for characters “Y” and “v” or the turtle movement string “tl” for characters “1” and “7” for “Gulim” font type. Also, most of the punctuation characters have the same turtle movement string which is “t”. To solve this problem, the length of edges of characters are compared. In other words, when the turtle movement string is not enough to find a character, the patch count of the edges are used. To this aim, similarity field is created in the tables.

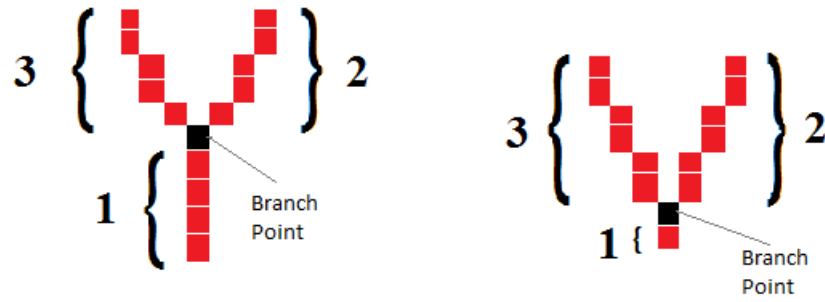


Figure 5.10: Edge comparison of character Y and v.

Edge No	Patch Count of The Edge (Y)	Patch Count of The Edge (v)
1	4	1
2	5	6
3	5	6

Table 5.2: The patch count of the edges for character Y and v.

In Figure 5.10, both character “Y” and “v” has three edges and in Table 5.2, the patch count of all edges for the characters are indicated. According to Table 5.2, the difference of edges is between edge number 1 and 2 for both characters. Using these edges, a condition can be created to determine if the character is “Y” or “v”. For example, in character “v”, the patch count of edge number 1 is smaller than half of the edge number 2, while this condition is never satisfied in character “Y”. So after a character is matched, if the turtle movement string is “t*tt”, the patch counts of edge number

1 and 2 are compared. If edge number 1 is smaller than half of the edge number 2, the character is recognized as “v” and “#” is written to the similarity column of character “v”. If not, the character is “Y” and similarity column of the “Y” is remained as null. This method is applied all characters which have same turtle movement string to find accurate character.

If the characters, which have the same turtle movement string, are the same letters such as “z” and “Z”, the method used in similarity column can not help to distinguish the characters because the rate between edges of characters are almost equal. In this case, the letter type of the characters are used which indicate if the letter is lower or upper. For example, the turtle movement string of “z” and “Z” is “lt” for “Vera” font type. Thus if the turtle movement string is “lt” and the letter height of the character is smaller than the line height, the letter type of the character is “lower” and so the character is “z”. If not, the letter type of the character is “upper” and so the character is “Z”.

Database Query from NetLogo

The MySQL extension for NetLogo establishes a connection to the databases and presents basic commands for queries. In the interface of the NetLogo model, a chooser is located to select the database which is either Gulim or Vera. When the user starts the turtle movement, the model directly connects to the selected database. After each character recognition is completed, according to the matching results four database queries are performed. These are:

- If the turtle movement string is unique in the selected database, only movement (turtle movement string) field is used in the query.
- If there are two characters matching with the turtle movement string in the database and the characters are not same letters, movement and similarity fields are used in the query.
- If there are two characters matching with the turtle movement string in the database and the characters are same letters (such as “s” and “S”), movement and type (letter type) fields are used in the query.
- If there are more than two characters matching with the turtle movement string in the database, movement, similarity and type fields are used in the query.

Figure 5.11 shows an example database query in NetLogo.

```

to database-query

set query "select name from "
set query (word query font " where movement = \"") )
set query (word query letter-string "\"") )
set query (word query " and type = \"") )
set query (word query letter-type "\"") )
set recognized-letter (mysql:executeQuery db query)

end

```

Figure 5.11: An example database query in NetLogo.

5.2.4 Displaying Results

In OCR applications, word recognition is an important step due to the effect on the accuracy rate. In this project, words are determined using line height and the distance between two characters. After a character is found from the database, the result is recorded to the string variable “text”, which holds recognized characters, in the NetLogo model. When the turtle comes to the start point of the next character, the difference between the start point of the new character and finish point of the last character gives the distance between these characters. If this distance is smaller than the line height, there is a word space between these characters and a space is added to the string “text”. Also, when the turtle arrives at end of a line, a space is added to the string “text” due to the new line. Figure 5.12 and Figure 5.13 show the NetLogo model extracting the text.

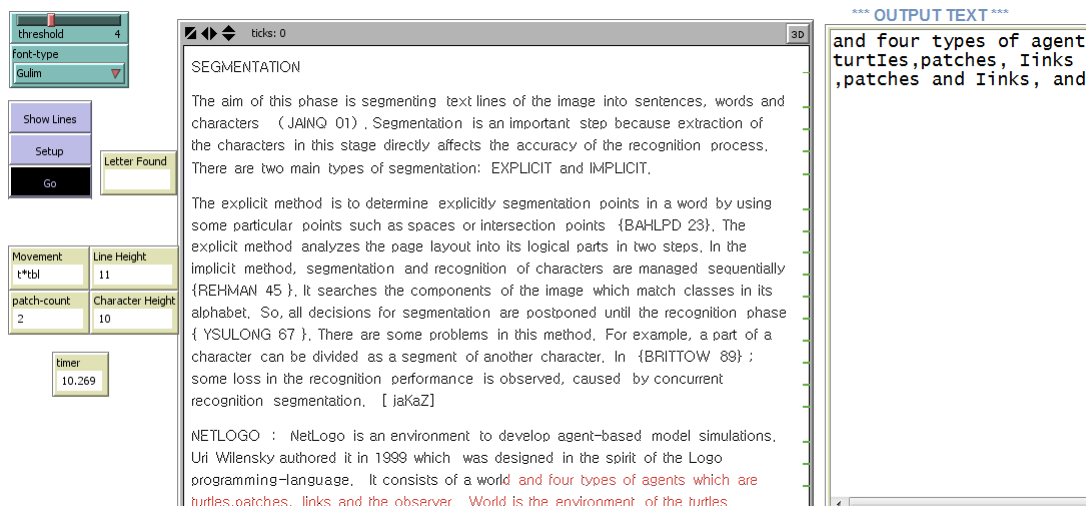


Figure 5.12: An example of the NetLogo model extracting the characters.

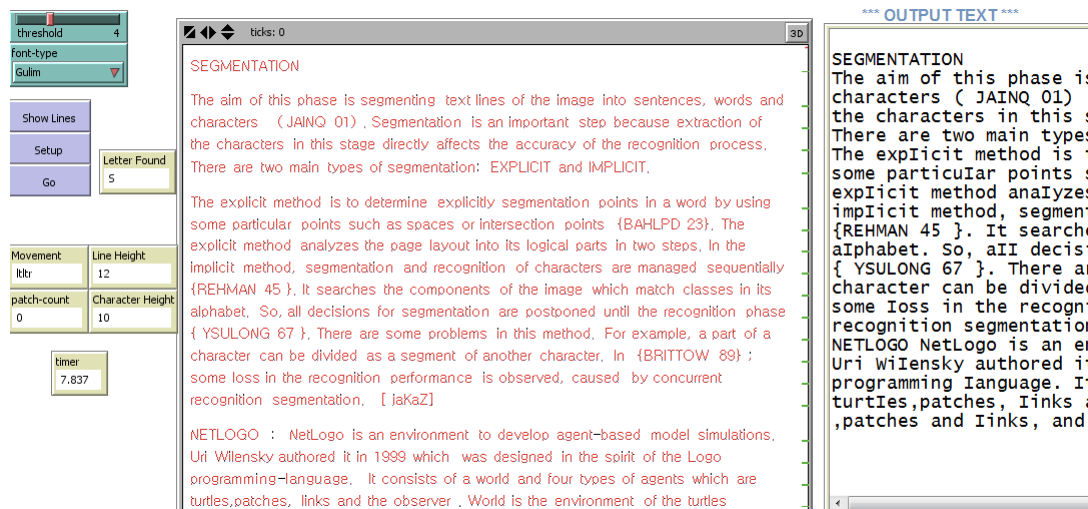


Figure 5.13: An example of the NetLogo model after the extraction of the characters completed.

The extracted characters are simultaneously shown in the output of NetLogo model. Also, there are some monitors on the interface to observe patch count, letter height, line height and turtle movement string step by step. When the turtle completes movement over characters, the recognized text is written to the file “result.txt”. Java reads the text from “result.txt” and uses this to calculate accuracy rate of the results.

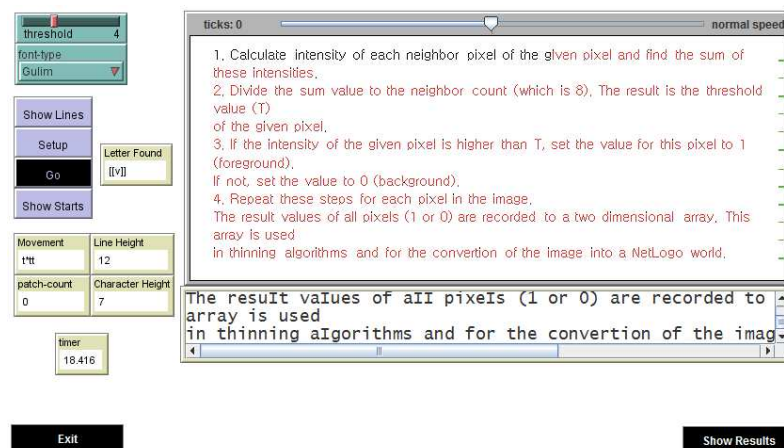


Figure 5.14: Displaying the NetLogo model on the Java Interface.

Also, the proposed NetLogo model is displayed on the Java interface as shown in Figure 5.14.

After the recognition task is completed by NetLogo, “Show Results” button is pressed to evaluate the results of recognition.

5.3 Summary

In this chapter, we have described how the image is transferred into NetLogo world and the methodology of the agent based approach for OCR. The image processed by Java is written in binary format to the “common.dat” document and this document is read by NetLogo to load the image into the world. If the item value in the document is 1, the corresponding patch in the world is colored to black. If the value is 0, the patch is colored as white.

The recognition process of the OCR implementation consists of four steps: line determination; character recognition; character matching from the database; and displaying results. In line determination, lines of the text are found using a threshold value and the turtle is located on each line once time. On the line, the turtle moves from right to left. If it finds a black patch ahead, the patch is accepted as start point of a character and the turtle starts the character recognition.

The turtle always researches for a black neighbor of the located patch and moves onto it to complete the character. In each step of the movement, it increases the patch count. When it comes to a corner, it increases the edge count and records the direction of movement. Also, the turtle can use branch points which allow the turtle to move on more than one patch in the next step. When there is no black neighbor of the located patch, movement on the character is completed and the located patch is recorded as finish point. At the end of the movement, we know start and finish points, edge count, patch count of each edge, the movement directions (turtle movement string) and the height of the character.

Two database tables are created to hold feature information of characters of two specific font types which are “Gulim” and “Bitstream Vera”. The tables hold letter type, turtle movement string, edge count and similarity information of the characters. The movement results of the turtle on a character are used in database queries to match the determined character against a character in the table. Firstly, the turtle movement string is searched in the table. If the turtle movement string is unique in the table, the character which has the same turtle movement string as in the table is recorded as the recognized character. If it is not unique, the letter type and similarity information of the determined character are used to find the correct character.

The recognized characters are concurrently displayed on the NetLogo interface and when the NetLogo model finishes recognition task, all recognized characters are written to the “results.txt”.

Chapter 6

Evaluation

This chapter discusses the effectiveness of the proposed OCR system. The first section presents the methods used in postprocessing of the recognized text. The second section describes the methods to evaluate performance of the OCR system. In section three, evaluation results of the system are analyzed while the section four compares this results with other OCR applications.

6.1 Postprocessing of Textual Output

The “result.txt” document, which is written by NetLogo after recognition of a text is completed, contains raw results of the recognized text. The recognized text can contain multiple spaces between words and some misrecognized characters. If there are multiple spaces between words instead of one space, each space is accepted as a character and reduces the accuracy rate even though for many applications the exact spacing is not required. Likewise the multiple spaces, misrecognized characters also affect the accuracy rate of the system. To improve performance of the system, some postprocessing methods are applied to the raw text. These methods can improve significantly the overall accuracy of the OCR system. They are explained as following.

- Multiple spaces can occur between two characters or in line breaks of the text. All multiple spaces in the recognized text are replaced with one space in Java.
- The characters “l” and “I” for “Gulim” font type were misrecognized in our system. They could not be separated since both of them have same letter type, turtle movement string and letter height. The only way to separate this letters is a word correction method. Various methods have been published for text correction of OCR output. The text correction method of Teahan et al. [47] has been implemented in our system to perform word correction. The method uses a Viterbi based

dynamic programming algorithms to find the most probable text string for possible corrected text strings based on a language model. This was implemented using the Text Mining Toolkit which allows for alternative spelling corrections to be performed and the best one is selected using the most probable string. For our system, two corrections of the letters “l” to “I” and vice versa were the only corrections needed but led to a significant improvement of performance.

Table 6.1 shows how the accuracy improves for a sample recognized text, which is written in “Gulim” font type, using the postprocessing methods shown in the table.

The method	Acc. (%)
Remove all spaces between characters	82
Leave space between words and leave line breaks	91
Apply word correction	95
Reduce all words spaces to one space	97
Replace line breaks with a single space	99.61

Table 6.1: An example of accuracy changing of a recognized text.

6.2 Evaluation of the Textual Output

The OCR applications are evaluated based on the accuracy rate of the results, time to extract the text, written type of the text (handwritten or typewritten), size of the text and paper quality. The OCR system proposed in this study is experimented on ten different texts. These texts are in different sizes and do not include any character which is not contained by the databases. The texts are in good quality papers and five of them in “Gulim” font type and 11 font size. The other texts are in “Vera” font type and 16 font size.

The accuracy rate of the extracted text from the image is calculated using Levenshtein distance metric [18]. The distance metric compares two strings and calculates their similarity rate (see section 2.6 for further details). In this project, this method is implemented in Java. Java reads the extracted text from the “result.txt” document and compares it with the original text, which is loaded by the user, using the distance method. The result of the distance metric gives the accuracy rate of the recognized text. Time to extract the texts from the images is calculated in NetLogo using a timer. Also, the threshold value on the interface of the model, which is used in line determination, is selected as 4.

The accuracy rate, character count of the original text and character count of the recognized text are displayed on the Java interface as shown in Figure 6.1. Also, the original text and the recognized text are displayed on the interface.

ORIGINAL TEXT	RECOGNISED TEXT	~ EVALUATION ~	
<p>SEGMENTATION</p> <p>The aim of this phase is segmenting text lines of the image into sentences, words and characters (JAINQ 01) . Segmentation is an important step because extraction of the characters in this stage directly affects the accuracy of the recognition process. There are two main types of segmentation: EXPLICIT and IMPLICIT.</p> <p>The explicit method is to determine explicitly segmentation points in a word by using some particular points such as spaces or intersection points {BAHLPOD 23}. The explicit method analyzes the page layout into its logical parts in two steps. In the implicit method, segmentation and recognition of characters are managed sequentially {REHMAN 45}. It searches the components of the image which match classes in its alphabet. So, all decisions for segmentation are postponed until the recognition phase {YSULONG 67}. There are some problems in this method. For example, a part of a character can be divided as a segment of another character. In {BRITTOW 89}; some loss in the recognition performance is observed, caused by concurrent recognition segmentation. [jaKaZ]</p> <p>NETLOGO : NetLogo is an environment to develop agent-based model simulations. Uri Wilensky authored it in 1999 which was designed in the spirit of the Logo programming-language. It consists of a world and four types of agents which are turtles,patches</p>	<p>SEGMENTATION</p> <p>The aim of this phase is segmenting text lines of the characters (JAINQ 01) . Segmentation is an important the characters in this stage directly affects the accuracy. There are two main types of segmentation EXPLICIT and IMPLICIT. The explicit method is to determine explicitly segmentation points such as spaces or intersection points. The explicit method analyzes the page layout into its logical parts in two steps. In the implicit method, segmentation and recognition of characters are managed sequentially {REHMAN 45}. It searches the components of the image which match classes in its alphabet. So, all decisions for segmentation are postponed until the recognition phase {YSULONG 67}. There are some problems in this method. For example, a part of a character can be divided as a segment of another character. In {BRITTOW 89}; some loss in the recognition performance is observed, caused by concurrent recognition segmentation. [jaKaZ]</p> <p>NETLOGO NetLogo is an environment to develop agent-based model simulations. Uri Wilensky authored it in 1999 which was designed in the spirit of the Logo programming-language. It consists of a world and four types of agents which are turtles,patches, links and the observer . World is the environment, patches and links, and it can be 2D or 3D. (FAGSE</p>	<p>Accuracy Rate (%) :</p>	<p>0.972</p>
		<p>Total Character :</p>	<p>1476</p>
		<p>Recognised Character :</p>	<p>1470</p>

Figure 6.1: An example of evaluation of recognized text

6.3 Analysis of the Evaluation Results

Table 6.2 and Table 6.3 show performance results of the OCR implementation for different texts. In this tables, dimensions of the images of the texts, the accuracy rates without word correction, the accuracy rates after word correction (only for “Gulim” font), character count in the original texts and in the recognized texts, and time to perform recognition are indicated.

No	Img. (wid.×heig.)	Acc. without Corr.(%)	Acc. with Corr.(%)	Char. (Org.)	Char. (Rec.)	Time (second)
1	633 × 261	95.6	99.0	600	600	16.472
2	624 × 496	97.3	99.6	1476	1470	29.085
3	673 × 713	96.9	99.4	2522	2521	47.719
4	674 × 1194	95.6	98.2	4390	4391	84.465
5	683 × 1478	96.8	99.7	4918	4915	95.68

Table 6.2: OCR implementation results of different texts in Gulim font type.

No	Img. (wid.×heig.)	Acc. without Corr.(%)	Char. (Org.)	Char. (Rec.)	Time (second)
1	631 × 330	99.3	442	440	7.664
2	681 × 892	99.6	1185	1183	23.932
3	674 × 1127	98.1	1540	1563	27.243
4	671 × 955	99.0	1594	1585	25.791
5	650 × 1642	99.2	2729	2723	45.558

Table 6.3: OCR implementation results of different texts in Bitstream Vera font type.

In this project, all characters are accurately recognized except “l” and “I” characters for “Gulim” font type. So word correction method is used for the texts in “Gulim” font type. According to Table 6.2, the accuracy rate of the texts increased after word correction implementation. However, in Table 6.3, word correction method is not required because all characters can be recognized. The accuracy results of the texts are quite high in both tables. But both of them can miss some spaces between words and so the accuracy rate is never 100%.

6.4 Comparison with Other OCR Systems

Comparing performance of the OCR systems is not easy due to the different texts and font types. Each OCR method has different advantages and disadvantages related to the written type of the texts (handwritten or typewritten), image quality and the algorithms adopted for image processing. Table

6.4 shows an experiment results to compare accuracy rate of six different commercial OCR devices in 1992 [41]. The text used in this experiment does not contain tables, figures and mathematical equations. The text is in a good quality paper, and consists of 132 pages and 278.786 characters.

OCR Device	Errors	Acc. (%)
Caere OmniPage Professional	8841	96.83
Calera RS 9000	3709	98.67
ExperVision TypeReader	6318	97.73
Kurzweil 5200	4716	98.31
Recognita Plus	11282	95.95
Toshiba ExpressReader	12169	95.64
ISRI Voting Algorithm	1867	99.33

Table 6.4: Accuracy comparison of some commercial OCR devices in 1992 [41].

Today, there are commercially-available OCR devices whose accuracy rates are almost 99 % in good quality papers. However, there is no OCR technology with a hundreded percent accuracy rate. Some of the texts, which are used to evaluate the OCR system proposed in this project, have been used to evaluate another OCR application which is provided online at <http://www.free-online-ocr.com> to have a comparison of our system with another OCR system. The results of this comparison are introduced in Table 6.5.

No(Font)	Char.(Org.)	Online OCR System			Our System		
		Char. (Rec.)	Acc.(%)	Time (second)	Char. (Rec.)	Acc.(%)	Time (second)
2(Vera)	1185	1182	98.9	11.861	1183	99.6	23.932
3(Vera)	1540	1734	86.3	10.424	1563	98.1	27.243
5(Vera)	2729	3579	67.1	18.300	2723	99.2	45.558
2(Gulim)	1476	1474	97.8	8.883	1470	99.6	29.085
3(Gulim)	2522	2536	97.7	13.082	2521	99.4	47.719
5(Gulim)	4918	4930	98.3	39.127	4915	99.7	95.680

Table 6.5: Comparison results of our system with an OCR application at <http://www.free-online-ocr.com> web site.

In the online OCR system, some characters were misrecognized in some places and some spaces between characters were missed. The misrecognized characters are as “.” with “,” , “l” with “/” and “1” , “i” with “j” , “r” with “f” , “a” with “s” , “s” with “5” and “g” with “9”. According to Table 6.5, the accuracy rate of the online OCR system for “Gulim” font type is more consistent than “Vera” font and over 97%. For “Vera” font, the accuracy rate is low for large size image which is number 5 in online OCR System, while our system reaches 99.2 % accuracy rate for the same image. The accuracy rate of our system is consistent for all images and more than 98 %. As a result, it is observed that the proposed OCR system in this dissertation has higher accuracy rates than the online OCR application for the tested texts. Nevertheless, our system spends longer time than the online system to perform the recognition process.

6.5 Summary

In this chapter, we have analyzed performance of the proposed OCR system and compared it with other OCR applications. To evaluate the performance, the accuracy rate of the system is calculated using Levenshtein distance method. Also, time to recognize the text is calculated by NetLogo with the help of a timer. The OCR system is experimented on ten different texts which are in particular fonts and sizes. The system recognized all characters accurately except “l” and “I” in “Gulim” font type and so a word correction method was used to correct them. The accuracy rate of the system was between 98 - 99.7 %. It could not reach 100 % accuracy rate because it missed some spaces between characters.

Also, the proposed system is compared with another online OCR application. The texts used for experiments to evaluate the proposed OCR system were also used on the online OCR application. The accuracy of the proposed system was higher then the online application as the online application misrecognized some words. However, the online application was faster than the proposed system particularly for large size images such as number 5 in “Vera” font type.

Chapter 7

Conclusions and Future Work

In this chapter, we summarize the project and then discuss the contributions of the project and future research directions.

7.1 Summary of the Dissertation

The aim of this project was to utilize an agent based approach for the OCR task. To this aim, an OCR system was developed using Java, NetLogo and MySQL environments. NetLogo provided the agent based environment for character recognition and MySQL supported the character recognition task of NetLogo by containing features of the characters. However, NetLogo was inadequate to create the interface of the OCR system and so Java was implemented to the system.

The OCR system was comprised of three main processes: image processing, character recognition and evaluation. The image processing step was conducted by Java to extract relevant textual information from the image, and convolution filters, thresholding methods and thinning algorithms were applied to the image according to choices of the user. Also, a new thinning algorithm was developed in this step. Thanks to the new thinning algorithm, the important details of the characters were preserved and only spurious parts of the characters were cleaned from the image. When the image processing was completed, pixel information of the processed image was written to the “common.dat” document and Java run the NetLogo model for recognition of the characters.

In the character recognition step, the processed image was transferred to the NetLogo world using the “common.dat” document. Then lines of the text were determined and a turtle was charged for moving on the characters. The turtle processed each line one at a time from right to left. While it was moving on a line, it determined the characters on the line and moved onto the characters

using the character recognition algorithm. When the turtle completed movement on a character, results of the movement were matched from the MySQL database to find the most similar character. The database contained the features of the characters which are type, edge count, movement (turtle movement string) and similarity. And the database was created for two specific font types which were “Gulim” and “Bitstream Vera”. This fonts were selected due to the their untouched character styles. When the turtle finished recognition of the whole image, the recognized text was recorded to the “result.txt”. The process flow of the character recognition is shown in Figure 7.1.

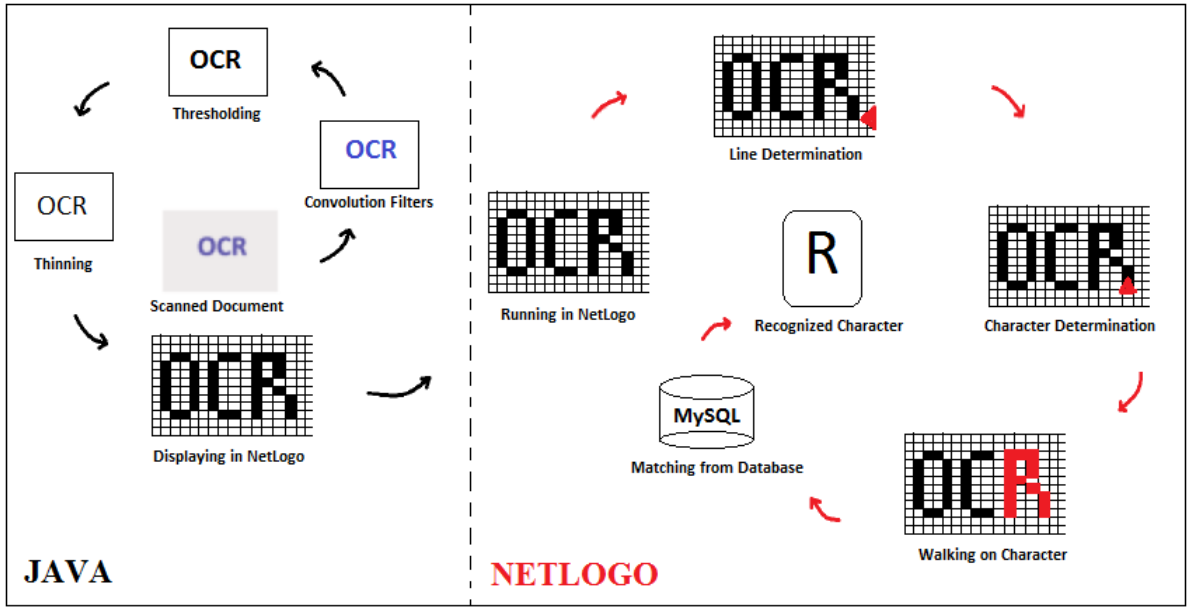


Figure 7.1: The process flow of the character recognition in the OCR implementation.

In the evaluation step, Java read the recognized text from the “result.txt” document to find accuracy rate of the text. To improve the accuracy rate, postprocessing algorithms, which are removing multiple spaces and word correction methods, were applied to the recognized text. Then the accuracy rate was calculated using the Levenshtein distance method.

The proposed OCR system in this dissertation was tested on different texts and the accuracy rate was between 98.1 - 99.7 % for this texts. Also, our system was compared with an online OCR application. Our system could reach higher accuracy rates than the online system, but the online system completed the recognition process in a shorter time.

7.2 Contributions

OCR is simply the process of reading characters from an image document. The methodology behind the OCR task is quite complicated. The OCR systems developed up to now are constructed on complex mathematical calculations and long process flow with many substeps. Also, most of the OCR systems extract the characters based on estimation rather than finding exact results and so require the support of word correction methods to increase their accuracy rates.

The contributions of the agent based approach for OCR are outlined as follows:

- In this project, we introduced an entirely new approach for OCR.
- OCR systems generally require word correction methods due to the misrecognized words. In our approach, there was no misrecognized characters in “Vera” font type and only the characters “l” and ‘I’ could not be recognized for “Gulim” font type. Thus the necessity of a word correction method was removed for “Vera” font type and reduced for “Gulim” font type.

Also, recognition of some similar characters such as “g” and “q”, “o” and “0” have been common problems of the OCR systems. Our approach can accurately separate these characters by walking over the lines of the characters and using pixel counts of the edges.

- Thanks to the consistency of the character recognition algorithm and exactly recognized characters, the overall accuracy rate of the OCR implementation was between 98.1 % and 99.7%, while the accuracy rate was between 67.1 % and 98.9 % for the same texts using an online OCR application.
- This approach can also be used for recognition of the alphabets which consist of dotted characters. For example, there are some characters in Turkish alphabet whose differences are only their dots such as “c” and “ç”, “s” and “ş”, “o” and “ö”, “u” and “ü”. This approach can easily detect these dots, their locations and connections to the edges and separate the characters.
- Our approach can also be adopted to develop an OCR system for hieroglyph writing. The hieroglyph writing is a form of picture writing and consists of symbols. The agent based approach can present a robust solution for recognition of hieroglyph scripts with the help of the logic of following lines of the shape.

The contributions of using NetLogo for implementation of the agent based approach are:

- Each step of the turtle and results of its steps were visualized concurrently thanks to the interface components of the NetLogo model. This advantage of NetLogo made it easier to

determine the weaknesses of the character recognition algorithm and observe the recognition process in a stepwise manner.

- The complexity of the character recognition task was reduced by merging some steps of the OCR. Recognition of the characters and words were conducted together. While the turtle was recognizing the characters, it also recorded which characters comprise a word. Thus when the turtle completed character recognition, words of the text were also able to be recognized. Further processing was not required for word recognition.

Also, the new thinning algorithm proposed in this project has made one of the most important contribution to implement the agent based approach for OCR. Since the character recognition was provided by following lines of the characters, if there was a gap or untouched edges on a character, the turtle would stop movement on the character. So the connectivity between edges of a character was an important factor for character recognition. The new thinning algorithm preserved this connectivity while some of the common thinning algorithms damaged it and cleaned the required parts of the characters.

7.3 Discussion

Beside the contributions of the OCR system, there are some weaknesses of the system in recognition of the text. These weaknesses are discussed the below.

Turtle Movement String: The major weakness of the recognition algorithm is limitations on the turtle movement string of characters. These limitations are stated as following:

- Characters are recognized matching their turtle movement strings against a record in the database. The turtle movement string holds the directions in the order used by turtle. This order depends on the start point of the character. If the start point changes, the turtle movement string would be different and would not match with records from the database.
- If the font size of a character is changed, the pixel locations on the thinned version of the character can be different. Because of this location changing of the pixels, different sizes of the same characters can have different turtle movement strings. Thus the turtle movement string may not match with the expected character in the database.

Spurious Parts: A spurious pixel between two characters can connect the characters and so the turtle accepts two characters as a single character. In Figure 7.2, the spurious pixel between the

characters “i” and “s” merges edges of these characters. Thus the turtle starts movement from “s” and completes the movement after moving to “i”.



Figure 7.2: An example of spurious pixels between two characters.

In the NetLogo model, a method is developed to recognize spurious parts of character “t” and “f”. But this method is specific to these characters and there is no general measure for spurious parts of all characters.

Gaps: If there is a gap on the edge of a character, the turtle completes the movement on the character before the gap and so can not move to the remaining parts of the characters. Figure 7.3 shows an example of a gap for the character “K”.

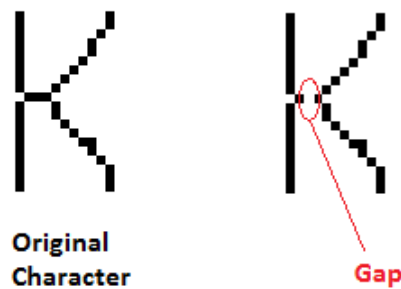


Figure 7.3: An example of gap on the character.

Time: The recognition task is conducted by only one turtle. The NetLogo model can complete recognition of an image, whose dimensions are 673×713 and consists of 2522 characters, in almost one minute. This time is acceptable but in larger images, recognition takes longer and affects performance of the OCR implementation.

7.4 Future Work

This project introduces a basic OCR system which uses an agent based approach. The OCR system successfully extracts the characters from the image with the help of agent based approach and

emphasizes possible advantages of NetLogo with its high accuracy rate. If the weaknesses of the proposed OCR system are removed, the new approach can present more robust solutions in character recognition.

The major weakness of the system was the turtle movement string. Instead of depending on start points of the characters and the order of the moving directions, it would be interesting to create a method which is independent from start point and can give the same result wherever the turtle starts its movement. Thus, the OCR system can work on all texts regardless of writing styles of the texts.

The spurious parts between characters was another weakness of the recognition algorithm. To get rid of this problem, the turtle can connect to the database at the end of each edge of a character and match the current turtle movement string of the character in the database. If the next change of direction will create a turtle movement string which does not match any record in the database, the turtle will be prevented from moving onto the new direction. And so the pixel at the end of the edge will be accepted as a spurious part of the character. It will be worthwhile adopting such a method to detect spurious parts of characters.

Also, the gaps on the characters can be completed by controlling the pixels after the gaps. For example, when the turtle comes to the pixel before a gap, it will check if the character is continuing after the gap. If the character continues after the gap, the gap will be colored black and so the turtle will continue the movement on the character.

Finally, the NetLogo model can complete the recognition task in shorter time by using multiple agents. One turtle for each line of the text can be charged for recognition instead of single turtle for the whole text. The turtles are run simultaneously and so significant time will be saved to complete the process.

Bibliography

- [1] A.N. Akansu, W.A. Serdijn, and I.W. Selesnick. Wavelet Transforms in Signal Processing: A Review of Emerging Applications. *Physical Communication, Elsevier*, 3(1):1–18, 2010.
- [2] N. Arica and F. T. Yarman Vural. One Dimensional Representation of Two Dimensional Information For HMM Based Handwritten Recognition. *Patten Recognition Letters*, 21(6):583–592, 2000.
- [3] L. Bahl, F. Jelinek, and R. Mercer. A Maximum Likelihood Approach to Speech Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2):179–190, 1983.
- [4] R. G. Baldwin. Using the Java 2D ConvolveOp Filter Class to Process Images. <http://www.developer.com/java/other/article.php/3696676/Using-the-Java-2D-ConvolveOp-Filter-Class-to-Process-Images.htm>, 2007. [Online; accessed 15-July-2012].
- [5] A. S. Britto, R. Sabourin, F. Bortolozzi, and C. Y. Suen. An Enhanced HMM Topology in an LBA Framework for the Recognition of Handwritten Numeral Strings. *Proceedings of the International Conference on Advances in Pattern Recognition*, pages 105–114, 2001.
- [6] R. G. Casey and E. Lecolinet. A Survey of Methods and Strategies in Character Segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 18(7):690–706, 1996.
- [7] K.R. Castleman. *Digital Image Processing*. Prentice-Hall, NJ, 2 edition, 1996.
- [8] M. Cheriet, N. Kharna, C. Liu, and C.Y. Suen. *Character Recognition Systems: A Guide for Students and Practitioners*. John Wiley & Sons, Inc., NJ, 2007.
- [9] Datapro Research Corporation. *All About Optical Readers*. Datapro Reports, NJ, 1989.

- [10] V. J. Dongre and V. H. Mankar. A review of Research on Devnagari Character Recognition. *International Journal of Computer Applications*, 12(2):8–15, 2010.
- [11] L. Eikvil. OCR-Optical Character Recognition. <http://www.nr.no/~eikvil/OCR.pdf>, 1993. [Online; accessed 24-June-2012].
- [12] K. Engel, Hadwiger, M., Kniss, J. M., Lefohn, A. E., C. R. Salama, and D. Weiskopf. *Real-time volume graphics*. CRC Press, Los Angeles, CA, 2006.
- [13] R. Fisher, S.Perkins, A. Walker, and E. Wolfart. Digital Filters. <http://homepages.inf.ed.ac.uk/rbf/HIPR2/filtops.htm>, 2003. [Online; accessed 13-June-2012].
- [14] J. Flusser and T. Suk. Pattern recognition by affine moment invariants. *Pattern Recognition*, 26(1):167–174, 1993.
- [15] Association for Automatic Identification and Mobility Inc. Optical Character Recognition (OCR). <http://www.aimglobal.org/technologies/othertechnologies/ocr.pdf>, 2000. [Online; accessed 23-July-2012].
- [16] R. C. Gonzalez. *Digital Image Processing*. Prentice Hall, USA, 2 edition, 2001.
- [17] A. Greensted. Otsu Thresholding. <http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>, 2010. [Online; accessed 7-August-2012].
- [18] R. Haldar and D. Mukhopadhyay. Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach. *CoRR*, 2011.
- [19] R. M. Haralick and L. G. Shapiro. *Computer and Robot Vision (Volume II)*. Prentice Hall, 2002.
- [20] S. Hjelmqvist. Fast, memory efficient Levenshtein algorithm. <http://www.codeproject.com/Articles/13525/Fast-memory-efficient-Levenshtein-algorithm>, 2006. [Online; accessed 11-August-2012].
- [21] Microscan Systems Inc. Understanding Optical Character Recognition. <http://www.microscan.com/en-us/technology/opticalcharacterrecognition.aspx>, 2011. [Online; accessed 10-August-2012].
- [22] L. C. Jain and B. Lazzerini. *Knowledge-Based Intelligent Techniques in Character Recognition*. CRC Press, London, 1999.

- [23] S. Knerr, V. Anisimov, O. Baret, N. Gorski, D. Price, and J. C. Simon. The A2iA Recognition System for Handwritten Checks. *Proceedings of Document Analysis Systems*, pages 431–494, 1996.
- [24] S. Knerr, L. Personnaz, and G. Dreyfus. Handwritten Digit Recognition by Neural Networks with Single-Layer Training. *IEEE Trans. on Neural Networks*, 3(6):962–968, 1992.
- [25] V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.
- [26] J. Lindsay. Spatial Filtering. <http://www.uoguelph.ca/~hydrogeo/Whitebox/Help/SpatialFilters.html>, 2009. [Online; accessed 9-July-2012].
- [27] H. Liu and X. Ding. Handwritten character recognition using gradient feature and quadratic classifier with multiple discrimination schemes. *Proceedings of the 8th International Conference on Document Analysis and Recognition*, pages 19–23, 2005.
- [28] Y. Ll. NetLogo Mysql Extension. <http://code.google.com/p/netlogo-mysql-extension/>, 2010. [Online; accessed 15-August-2012].
- [29] Y. Lu and M. Shridhar. Character Segmentation in Handwritten Words. *Pattern Recognition*, 29(1):77–96, 1996.
- [30] I. S. MacKenzie and R.W. Soukoreff. Text Entry For Mobile Computing Models and Methods, Theory and Praticce. *Human-Computer Interaction*, 17:147–198, 2002.
- [31] J. Mantas. An overview of character recognition methodologies. *Pattern Recognition*, 19(6):425–430, 1986.
- [32] P. M. Mather. *Computer Processing of Remotely Sensed Images*. John Wiley & Sons, West Sussex, 2004.
- [33] A. Mellouk and A. Chebira. *Machine Learning*. InTech, Crotia, 2009.
- [34] T. M. Mitchell. The Discipline of Machine Learning. <http://www.cs.cmu.edu/~tom/pubs/MachineLearning.pdf>, 2006. [Online; accessed 18-July-2012].
- [35] K. M. Mohiuddin and J. Mao. Optical Character Recognition. *Wiley Encyclopedia of Electrical and Electronics Engineering*, 15:226–236, 1999.

- [36] D. J. H. Moore. On the Medial Axis Function for Visual Patterns. *IEEE Transactions on Systems, Man and Cybernetics*, 4(4):396–399, 1974.
- [37] W. Niblack. *An Introduction to Digital Image Processing*. Prentice Hall, NJ, 1986.
- [38] N. Otsu. A Threshold Selection Method from Gray Level Histograms. *IEEE Trans. on Systems, Man and Cybernetics*, 9(1):62–66, 1979.
- [39] M. A. Ozdil and F. Y. Vural. Optical Character Recognition Without Segmentation. *Lecture Notes on Computer Science*, 1311:609–615, 1997.
- [40] A. Rehman, D. Mohamad, and G. Sulong. Implicit vs Explicit based Script Segmentation and Recognition: A Performance Comparison on Benchmark Database. *Open Problems Compt. Math.*, 2(3):353–361, 2009.
- [41] J. Kanai S. V. Rice and T. A. Nartker. A Report on the Accuracy of OCR Devices. *Symposium on Document Analysis and Information Retrieval*, 1992.
- [42] H. M. Schey. *DIV, Grad, Curl, and All That: An Informal Text on Vector Calculus*. W. W. Norton & Company, 3 edition, 1997.
- [43] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.
- [44] A. Smola and S. V. N. Vishwanathan. *Introduction to Machine Learning*. Cambridge University, UK, 2008.
- [45] G. Srikanthan, S. W. Lam, and S. N. Srihari. Gradient-Based Contour Encoding for Character Recognition. *Pattern Recognition*, 29(7):1147–1160, 1996.
- [46] G. Sulong, A. Rehman, and T. Saba. Improved Offline Connected Script Recognition Based on Hybrid Strategy. *Engineering Science and Technology*, 2(6):2010, 1603-1611.
- [47] W. J. Teahan, S. Inglis, J. G. Cleary, and G. Holmes. Correcting English Text Using PPM Models. *Data Compression Conference*, pages 289–298, 1998.
- [48] G. Vamvakas. Optical Character Recognition for Handwritten Characters. *Institute of Informatics and Telecommunications and Computational Intelligence Laboratory (CIL), Greece*.
- [49] N. Wiener. *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*. The MIT Press, New York, 1964.

- [50] I. Z. Yalniz and R. Manmatha. A Fast Alignment Scheme for Automatic OCR Evaluation of Books. *ICDAR'11 Proceedings of the 2011 International Conference on Document Analysis and Recognition*, pages 754–758, 2011.
- [51] T. J. Zhang and C. Y. Suen. A fast parallel algorithm for skeletonization digital pattern. *Comm. of ACM*, 27:236–239, 1984.
- [52] N. Zuech. *Understanding and Applying Machine Vision*. Taylor & Francis, 3 edition, 2000.

Appendix A

Pseudocode of Character Recognition

This is a pseudocode presentation of the character recognition method of the NetLogo model. The full source can be found on the attached CD.

Variables

- int posX = 0	- int patch-count = 0
- int posY = 0	- int edge = 0
- int startX = 0	- int line-height = 0
- int startY = 0	- int letter-height = 0
- int finishX = 0	- string letter-string = ""
- int finishY = 0	- string letter-type = ""
- int branch = 0	- list length-list

Procedure Research

```
move forward 1 step
IF color of patch ahead = black THEN
  Move-on-letter
ELSE
  Research
END IF
```

End Procedure

Procedure Move-on-letter

```
posX = x coordinate of the turtle
posY = y coordinate of the turtle
startX = posX
startY = posY
finishX = posX
finishY = posY
IF color of patch at (posX, posY + 1) = black THEN
  letter-string += "t"
  Move-top
  edge += 1
```

```

ELSE IF color of patch at (posX+1, posY) = black THEN
    letter-string += "r"
    Move-right
    edge += 1

ELSE IF color of patch at (posX-1, posY) = black THEN
    letter-string += "l"
    Move-left
    edge += 1

ELSE IF color of patch at (posX, posY-1) = black THEN
    letter-string += "b"
    Move-bottom
    edge += 1

END IF
Record-length
Find-Character
End Procedure

```

```

Procedure Move-top
    Move on coordinates (posX, posY+1)
    posX = x coordinate of the turtle
    posY = y coordinate of the turtle
    set color of patch-here red
    patch-count += 1
    Check-branch
    IF color of patch at (posX, posY + 1) = black THEN
        Move-top
    ELSE IF color of patch at (posX+1, posY) = black THEN
        letter-string += "r"
        Move-right
        Record-length
    ELSE IF color of patch at (posX-1, posY) = black THEN
        letter-string += "l"
        Move-left
        Record-length
    ELSE IF color of patch at (posX, posY-1) = black THEN
        letter-string += "b"
        Move-bottom
        Record-length
    END IF
End Procedure

```

Procedure Move-right

```
Move on coordinates (posX+1, posY)
posX = x coordinate of the turtle
posY = y coordinate of the turtle
set color of patch-here red
patch-count += 1
Check-branch
IF color of patch at (posX, posY + 1) = black THEN
    letter-string += "t"
    Move-top
    Record-length
ELSE IF color of patch at (posX+1, posY) = black THEN
    Move-right
ELSE IF color of patch at (posX-1, posY) = black THEN
    letter-string += "l"
    Move-left
    Record-length
ELSE IF color of patch at (posX, posY-1) = black THEN
    letter-string += "b"
    Move-bottom
    Record-length
END IF
```

End Procedure**Procedure Move-left**

```
Move on coordinates (posX-1, posY)
posX = x coordinate of the turtle
posY = y coordinate of the turtle
set color of patch-here red
patch-count += 1
Check-branch
IF color of patch at (posX, posY + 1) = black THEN
    letter-string += "t"
    Move-top
    Record-length
ELSE IF color of patch at (posX+1, posY) = black THEN
    letter-string += "r"
    Move-right
    Record-length
ELSE IF color of patch at (posX-1, posY) = black THEN
    Move-left
```

```

ELSE IF color of patch at (posX, posY-1) = black THEN
    letter-string += "b"
    Move-bottom
    Record-length
END IF
End Procedure

Procedure Move-bottom
    Move on coordinates (posX, posY-1)
    posX = x coordinate of the turtle
    posY = y coordinate of the turtle
    set color of patch-here red
    patch-count += 1
    Check-branch
    IF color of patch at (posX, posY + 1) = black THEN
        letter-string += "t"
        Move-top
        Record-length
    ELSE IF color of patch at (posX+1, posY) = black THEN
        letter-string += "r"
        Move-right
        Record-length
    ELSE IF color of patch at (posX-1, posY) = black THEN
        letter-string += "l"
        Move-left
        Record-length
    ELSE IF color of patch at (posX, posY-1) = black THEN
        Move-bottom
    END IF
End Procedure

Procedure Check-branch
    IF (black neighbor count of patch at (posX,posY) > 1 ) OR
        (black neighbor count of patch at (posX,posY) = 1
        and there is a corner) THEN
        letter-string += "*"
        branch = neighbor-count
        record coordinates of neighbors
        DO WHILE branch > 0
            Move on neighbor patch
            branch = branch - 1
        END DO
    END IF
End Procedure

```

```

Procedure Record-length
    IF posX > startX THEN
        startX = posX
    END IF
    IF posX < finishX THEN
        finishX = posX
    END IF
    IF (posY - startY) > line-height THEN
        line-height = (posY - startY)
    END IF
    IF (posY - startY) > letter-height THEN
        letter-height = (posY - startY)
    END IF
    edge += 1
    addlength-list patch-count
    patch-count = 0
End Procedure

Procedure Find-character
    IF letter-height < line-height THEN
        letter-type = "lower"
    ELSE
        letter-type = "upper"
    find from database where movement = letter-string
    and type = letter-type
    print database result
End Procedure

```

Figure A.1: Pseudocode of the Character Recognition Algorithm

Appendix B

UML Diagram

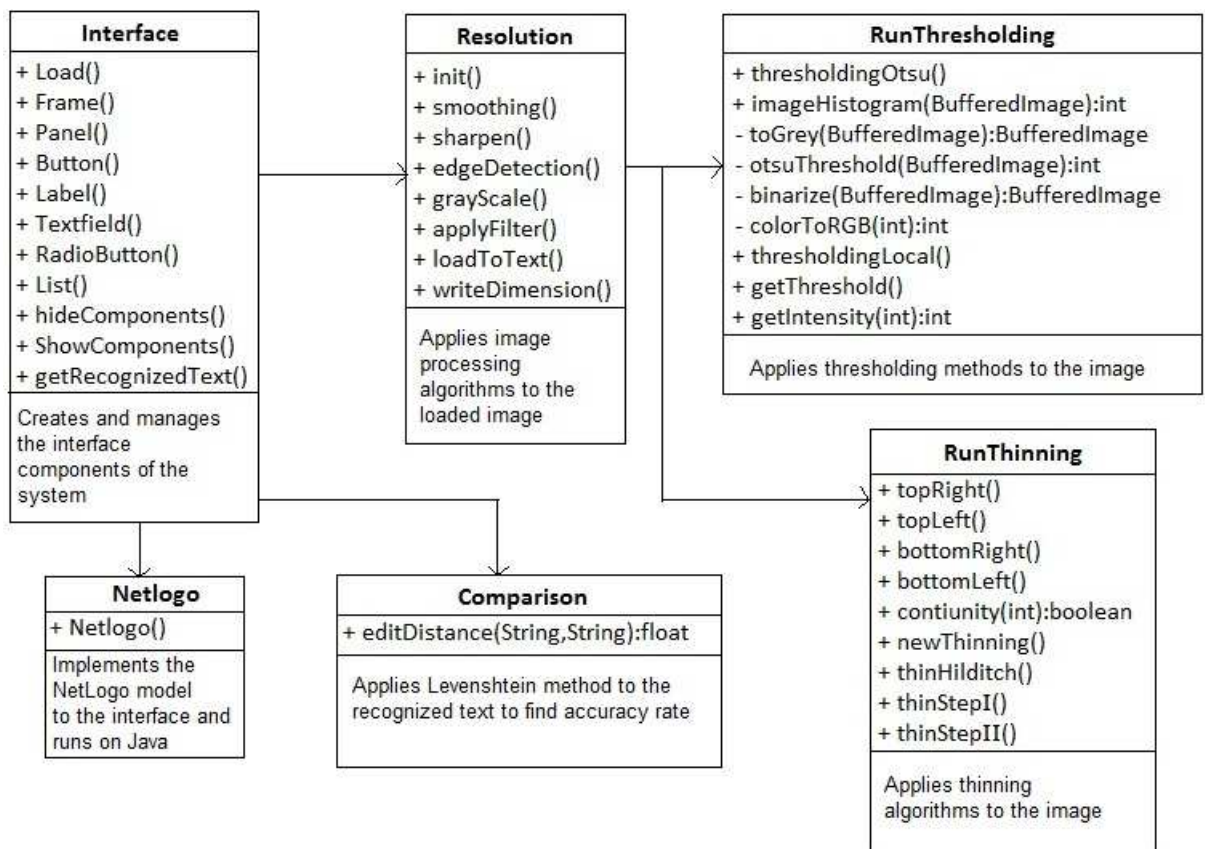


Figure B.1: UML Diagram for Java Classes