



Faculty of Science and Technology  
Department of Electrical Engineering and Computer Science

# Interface Development for Digitization of Documents Using OCR

BACHELOR'S THESIS IN COMPUTER SCIENCE

BY

CHRISTOFFER LINDLAND AND

FADUL ELWALID FADUL

UNDER THE SUPERVISION OF

PROF. KARL SKRETTING

UNIVERSITY OF STAVANGER

KRISTIAN STRÅBØ

AKER SOLUTIONS

# Abstract

The purpose of this thesis is to develop a semi-automated interface that uses Optical Character Recognition (OCR) routines to identify text-based information from a large volume of digitized drawings associated with the oil and gas industry. The identified information is presented in an appropriate interface for any necessary manual modification, with the target of improving the efficiency of maintaining large amounts of older documents. The thesis outlines the design of the interface and the implementation of Tesseract OCR engine, in combination with tailor-made functions and classes that leverage OpenCV to enhance the recognition process.

# Acknowledgements

We would like to express our deepest gratitude to our supervisor, Professor Karl Skretting, for his invaluable guidance and unwavering support throughout the course of our thesis. His expertise and insightful feedback have been instrumental in shaping the the direction of our research and have contributed significantly to our growth as developers.

We would also like to extend our sincere appreciation to our external supervisor, Kristian Stråbø, for his invaluable knowledge and expertise in the oil and gas industry. His insightful feedback and contribution to the design of the interface have been critical to the success of our project.

In addition, we are grateful to the staff at Aker Solutions, specifically the department at Jåttåvågen, for providing us with office space and academic feedback during the writing of our thesis. We would also like to express our gratitude to the faculty and staff of the Department of Electrical Engineering and Computer Science for providing us with necessary knowledge and expertise.

Finally, we would like to thank our families and friends for their unwavering support and encouragement throughout our academic journey. Without their love and support, this achievement would not have been possible.

# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgements</b>	<b>ii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Objective . . . . .	1
1.3 Significance . . . . .	2
1.4 Background and Preliminary Concepts . . . . .	2
1.5 Confidentiality Measures . . . . .	2
<b>2 Foundational Concepts and Techniques</b>	<b>3</b>
2.1 User Interface . . . . .	3
2.1.1 Graphical User Interface . . . . .	3
2.2 Image Processing . . . . .	6
2.2.1 Open Source Computer Vision Library . . . . .	7
2.2.2 Morphological Operations . . . . .	7
2.3 Optical Character Recognition . . . . .	9
2.3.1 Image Pre-processing Pipeline . . . . .	10
2.3.2 Text Recognition Pipeline . . . . .	17
2.3.3 Long Short-Term Memory . . . . .	18
2.3.4 Types of Optical Character Recognition Systems . . . . .	19
2.3.5 OCR Engine . . . . .	19
2.3.6 Configurations . . . . .	19

<b>3 Suggested Approach</b>	<b>21</b>
3.1 Design . . . . .	21
3.2 Implementation . . . . .	23
3.2.1 Import PDFs and Begin Processing . . . . .	23
3.2.2 Converting PDF . . . . .	26
3.2.3 Image Processor . . . . .	27
3.2.4 Hardware Extraction . . . . .	31
3.2.5 Saving to Excel and Cleanup . . . . .	35
3.3 Performance Experiment . . . . .	37
<b>4 Discussion and Conclusion</b>	<b>39</b>
4.1 Discussion . . . . .	39
4.2 Conclusion . . . . .	40

# Chapter 1

## Introduction

### 1.1 Background

Many older platforms in the oil and gas industry are still in operation, some of which date back to the 1970s [1], which means that there is a large amount of documentation that needs to be renewed. Aker Solutions has a project where there is a need to extract information from a considerable amount of old drawings, approximately 10.000 drawings in PDF format, with varying degrees of quality. Their goal is to simplify maintenance of these drawings and the important text-based information. This bachelor's thesis is therefore a collaboration effort between the University of Stavanger and the external company Aker Solutions, intended to tackle the challenges of extracting information from a vast amount of drawings. The collaboration is primarily with the department located at Jåttåvågen, where the external supervisor works closely with the developer team. The external supervisor provides constant feedback regarding the desired information that needs to be extracted from the drawings, as well as suggestions regarding the user interface. Given that the external supervisor will be the end-user of the final product, their input on the user interface is invaluable to the team in developing an effective solution to the problem at hand, as well as a product designed with the user's perspective in mind.

### 1.2 Objective

The objective is to develop a semi-automated interface in Python that uses Optical Character Recognition (OCR) routines to recognize text in the digitized drawings, and present the outcome in an appropriate interface, making it easily understandable to determine the recognized and unrecognized text. The interface will also enable the user with the functionality to manually modify and adjust the OCR outcome. Additionally, the ex-

ternal company requested the ability to save the extracted information as, i.e., an Excel spreadsheet. The additional request was therefore taken into consideration during the development process.

### 1.3 Significance

The significant importance of this collaboration project originates from the digitization transition within the oil and gas industry, including Aker Solutions, as it seeks to improve the efficiency of maintaining a large volum of older drawings or documents. Such an initiative is especially important for the oil and gas industry, where accurate and up-to-date information is critical for ensuring safety and compliance [2]. By enhancing the accessibility and accuracy of information through modernized technology, the project has the potential to reduce workload and high maintenance costs. In sum, this project presents significant potential for generating substantial benefits not only for Aker Solutions but also for the industry and the wider communities as they embrace the ongoing digital transformation.

### 1.4 Background and Preliminary Concepts

Given the project's objectives and goals, it is crucial to have a comprehensive understanding of Optical Character Recognition (OCR) techniques and their effective utilization. Equally important is the ability to create a user-friendly interface, which is essential for a successful product. Chapter 2 provides necessary background information covering the fundamental concepts related to OCR and user interface. Chapter 3 presents the approach taken to develop the final product, which include a detailed description of algorithms, and tools utilized to implement OCR techniques and the design and creation of the user-friendly interface.

### 1.5 Confidentiality Measures

Furthermore, it is crucial to highlight that throught this thesis, all images and drawings in Chapter 3 have been anonymized to safeguard the privacy and confidentiality of external company involved in this project.

# Chapter 2

## Foundational Concepts and Techniques

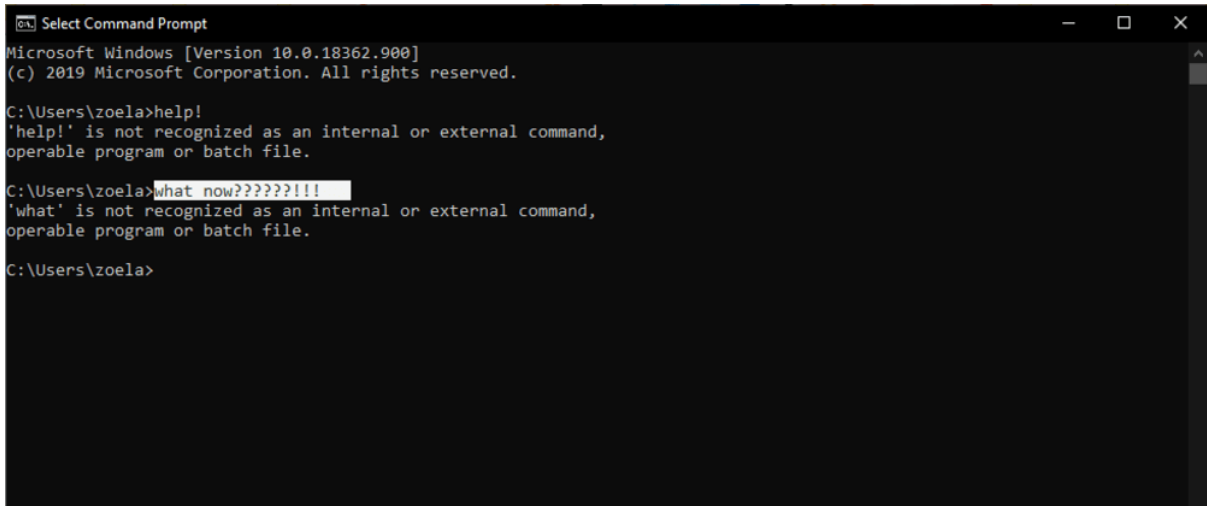
### 2.1 User Interface

The User Interface (UI) is a critical component of software systems [3]-[4], as it serves as a point of human-computer interaction as well as user interaction with a software application or a website. This interface is constructed in a series of layers that are designed to appeal to human senses, acting as a bridge that facilitates communication between the user and software. Typically, UI incorporates both input and output devices, including keyboards, mouse trackpads, microphones, touch screens, monitors, and speakers. For instance, everyday UI uses utilizes a combination of tactile input (keyboard and mouse) and visual auditory output (monitor and speakers). The design of a UI is critical for software usability and can have significant impact on the overall experience [5]. Therefore, it is essential to take into account the needs and preferences of the target audience when designing or developing a UI.

#### 2.1.1 Graphical User Interface

The graphical user interface (GUI) is a type of UI that primarily employs visual elements and graphical representations to present information and facilitate interaction between the user and software applications [6]. GUIs commonly use windows, icons, and menus to execute various commands, including file manipulation tasks. In contrast to text-based command-line interfaces (CLIs) such as MS-DOS or Unix-like shells, GUIs are typically considered more user-friendly, as they do not require the user to memorize commands or programming language. Figures 2.1 and 2.2 present visual examples of both CLI and GUI, respectively.





```
Select Command Prompt
Microsoft Windows [Version 10.0.18362.900]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\zoela>help!
'help!' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\zoela>what now?????!!!
'what' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\zoela>
```

Figure 2.1: Screenshot of Command Line Interface (CLI) running on Windows 10. Source: [7]

Examples of GUIs include popular operating systems such as Windows, MacOS, and Android, which enable users to execute commands via gestures or mouse movements without the need to enter any code [8]. Consequently, GUIs have become an ubiquitous feature of modern software applications, revolutionizing the way people interact with and utilize digital tools [9].

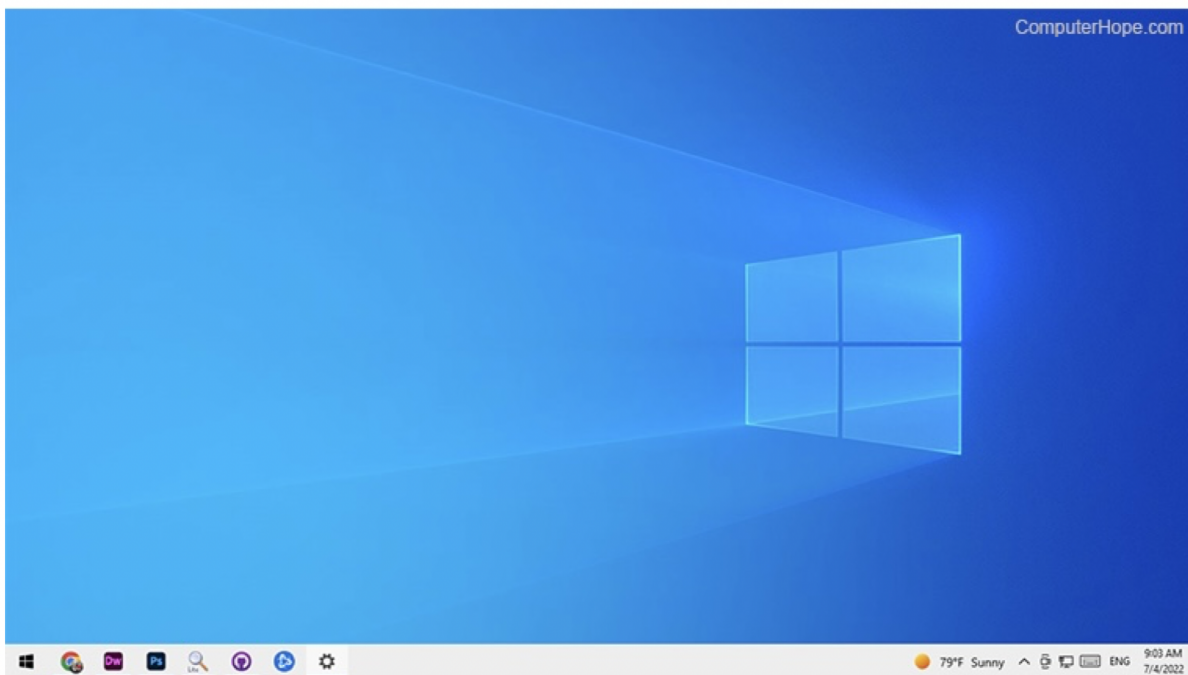


Figure 2.2: Example of a Graphical User Interface (GUI) from a Windows 10 desktop environment. Source: [6]

## Elements of a GUI

To enhance user interaction with software and optimize the user-friendliness of a GUI, various elements and objects are utilized. GUI elements can generally be classified into two categories: text and non-text (refer to Figure 2.3). Presented below is a few examples of GUI elements commonly used.

- Buttons: graphical objects that the user can click on to trigger an action.
- Icons: small graphical representation of files folders, or other objects.
- Text fields: areas where the user can input text or data.
- Checkboxes: small boxes that the user can select or deselect to indicate a choice.
- Radio button: small circles that the user can select to indicate a choice.
- Sliders: graphical objects that the use can slide back and forth to adjust a value or setting.
- Menu: list of options that the user can select from.
- Toolbars: graphical areas that contain buttons or icons representing frequently used commands or functions.
- Progress bars: graphical indicators that show the progress of a task or operation.

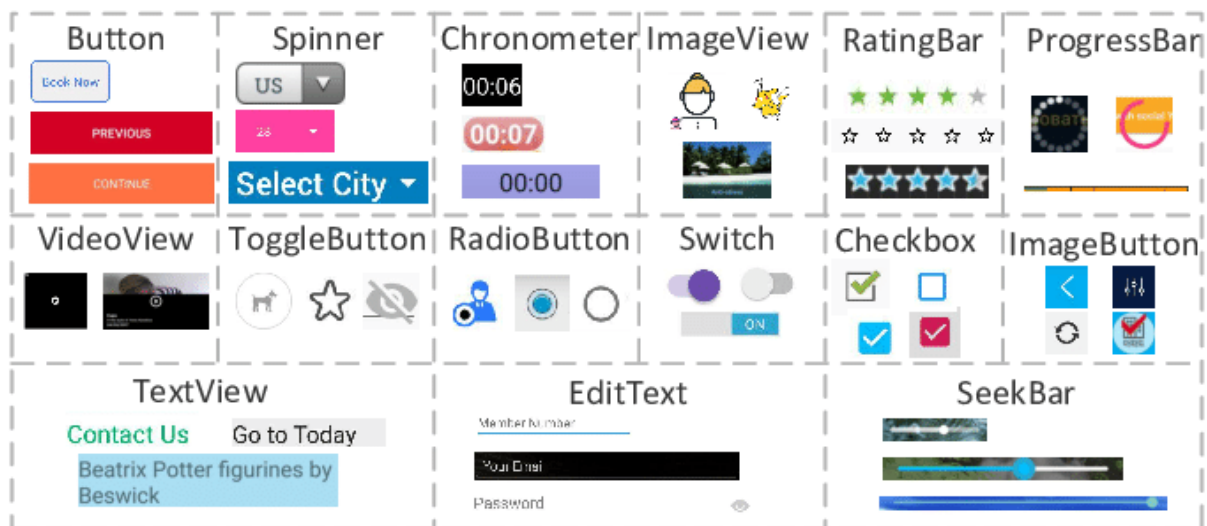


Figure 2.3: Examples of Android GUI elements. Source: [10]

## 2.2 Image Processing

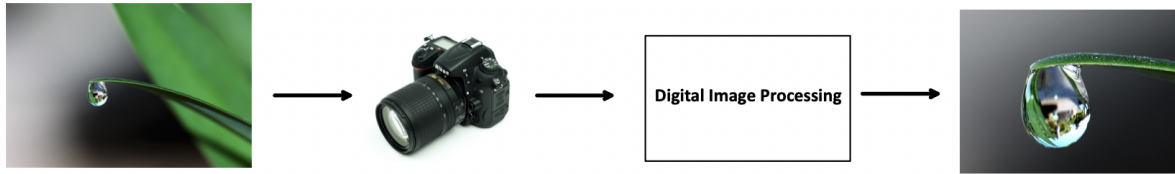


Figure 2.4: An image of a water drop captured by a camera and processed through a digital system. The system removes all extraneous details, focusing solely on the water drop while preserving the image quality.

Image processing is a set of computational techniques to enhance raw images captured through the use of various vision sensors [11], or extract useful information from the captured images. Essentially, it is a type of signal processing where the input is an image, and the output may be either an altered image or characteristics associated with the input image. The process consists of three essential steps, starting with importing the image using image acquisition tools [12]. Thereafter, analyzing and manipulating the image, and finally obtaining an output that may be an altered image or report based on image analysis [13].

There are two main types of methods used in image processing: analogue and digital image processing. This project focuses on digital image processing (DIP) techniques, which involve manipulating digital images through three general phases: pre-processing, enhancement, and display, which culminate in information extraction.

### Digital Image

A digital image is a two-dimensional representation of visual information in a digital format, composed of pixels arranged in columns and rows [14]. Each pixel represents a small portion of the image, and its intensity (or color) is stored as a binary code. Digital images can be manipulated using DIP techniques to manipulate these groups of bits or pixels to enhance image quality, create different perspectives, or extract information with the aid of computer algorithms.

Figure 2.5 demonstrates the impact of different color bit depths (or bits per pixel) on the visual quality of the image. The 1-bit image is known as a binary image which consists of only two colors, where each pixel is either black or white. The black pixels represent the foreground, and the white pixels represent the background [15].

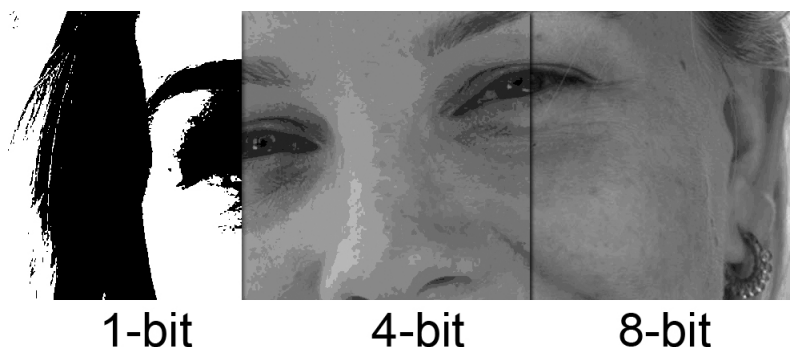


Figure 2.5: Original image is split into 1-bit, 4-bit, and 8-bit (grayscale) colors, representing 2, 16, and 256 colors, respectively. Source: [16]

### 2.2.1 Open Source Computer Vision Library

OpenCV, short for Open Source Computer Vision Library, is an open-source library of programming functions that provides a vast array of tools and algorithms for image processing [17]. Its functions include image filtering, feature detection, object detection, and more [18]. It supports multiple programming languages, such as C++, Python, and Java, which makes it a versatile and accessible tool for image processing tasks [18]. The wide range of algorithms and functions that OpenCV offers, coupled with its user-friendly interface, makes it the preferred choice for the present project as well as various image processing applications [17].

Overall, OpenCV is a powerful tool for image processing that offers a broad range of functions and algorithms to solve a diverse set of problems. Its open-source nature and versatility make it an excellent choice for researchers and developers alike.

### 2.2.2 Morphological Operations

Morphology encompass an extensive variety of image processing operations that process images based on shape [19], [20]. Morphological operations involves applying a structuring element to an input image, resulting in an output image of the same size. In essence, morphological operations determine the output pixel value based on a comparison between the corresponding input pixel and its neighboring pixels. Although the term "morphology" is typically used in biology to refer to the form and structure of animals and plants, in mathematical morphology, the same term is used to describe the process of extracting image components that are useful for representing region shape, boundaries, and more [21]. Erosion and dilation are fundamental morphological operations in image processing that are widely implemented in various applications [22], and are illustrated in Figures 2.7 and 2.8.

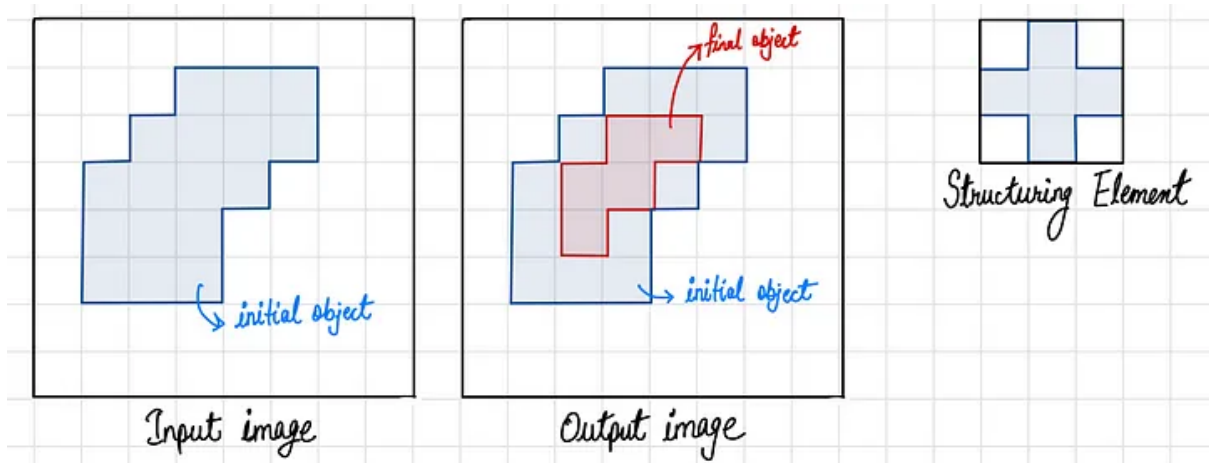


Figure 2.6: Erosion operation on an input image using a structuring element. Source: [21]

Morphology and Image Segmentation share a common ground in terms of image processing techniques. Morphology encompasses a range of techniques that can be applied as either pre-processing or post-processing steps. Implementing morphological operations as a pre-processing step refines the input data for Image Segmentation, while implementing it as a post-processing step can effectively eliminate any inaccuracies present in the segmented image. Moreover, morphological operations can provide valuable insights about the structural and shape features of the image, which are essential for subsequent image analysis tasks. Therefore, the integration of both Morphology and Image Segmentation can lead to more robust and accurate image processing algorithms [23]. For more comprehensive understanding of the process of Image Segmentation, readers should refer to Section 2.3.1 later in this thesis.

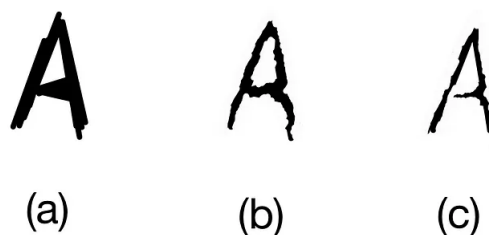


Figure 2.7: (a) Original image, (b) and (c) shows processed images after erosion using 3x3 and 5x5 structuring elements respectively. Source: [21]

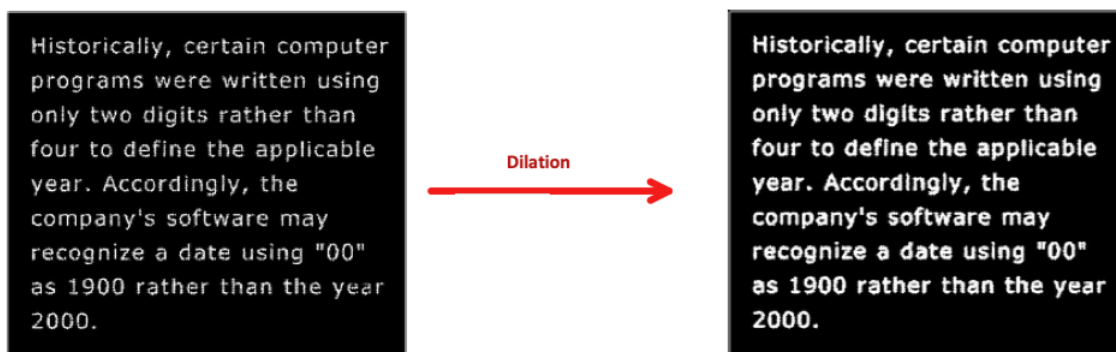


Figure 2.8: Image before dilation (left) with gaps in image text. Image after dilation with gaps within the text filled. Source: [22]

## 2.3 Optical Character Recognition

Optical Character Recognition (OCR) is a process that involves converting printed, type-written or handwritten text images into digital machine-encoded text [24, p. 19-20], [25]. OCR is used to recognize characters that are optically drawn or printed, making it possible to convert scanned documents or images into searchable and editable digital text [26]. The OCR algorithm consists of several major steps, beginning with the initial stage called image acquisition where data is fed into the system. The next step is preprocessing, where the image quality is improved using various techniques. The third step is character segmentation, which involves separating the characters that make up the image, and forms the initial stage of the OCR algorithm. Figure 2.10 shows the steps for OCR, and Figure 2.9 illustrates the workflow.

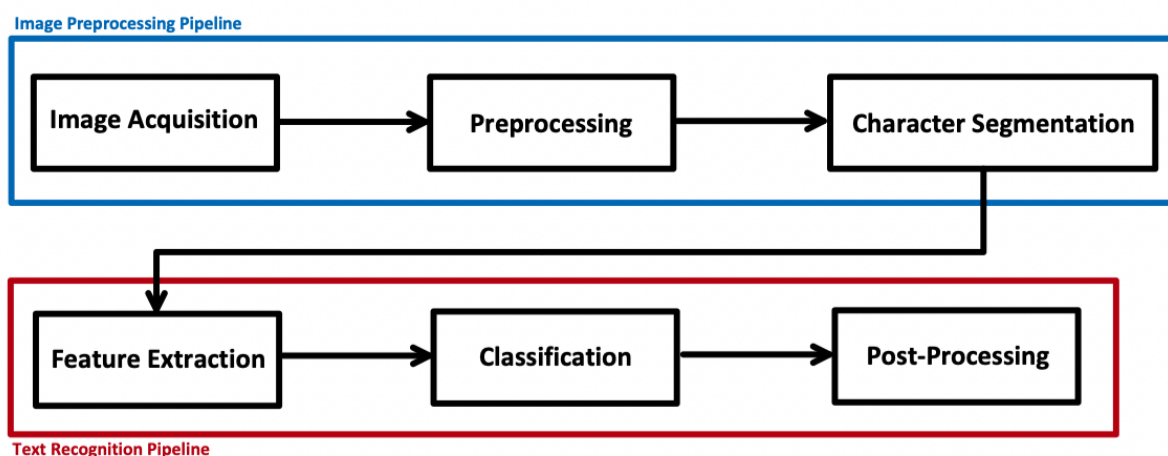


Figure 2.9: General model of a OCR process flow.

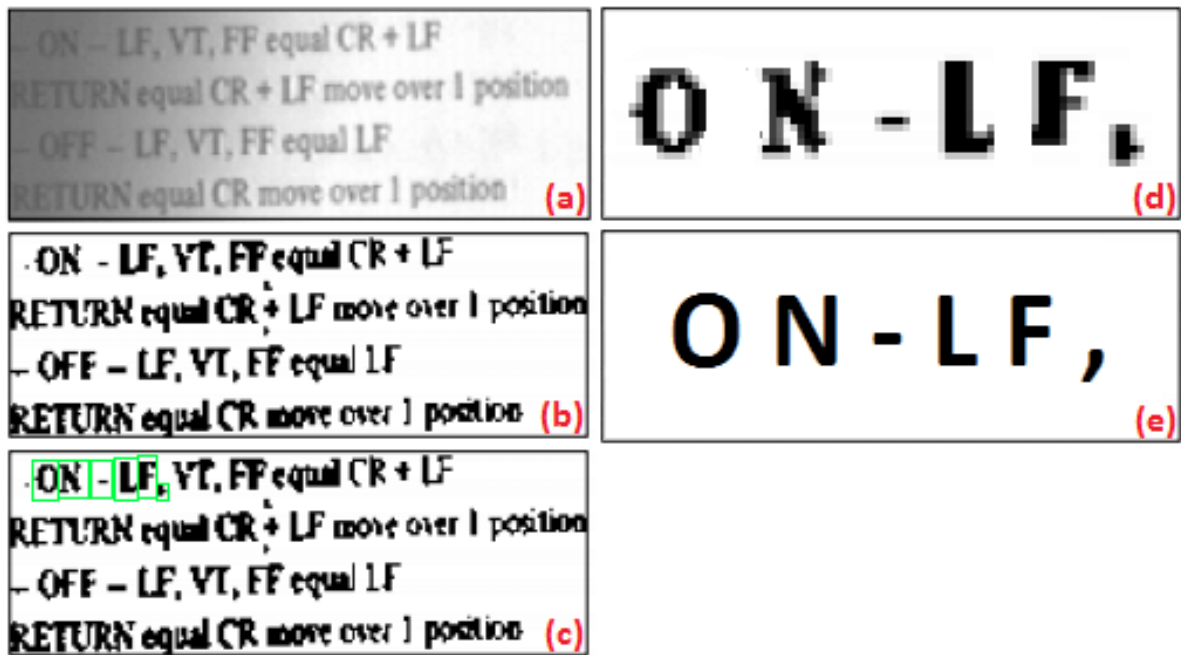


Figure 2.10: Steps for OCR. (a) Image acquisition. (b) Preprocessing. (c) Character segmentation. (d) Feature extraction. (e) Classification and post-processing. Source: [26]

Moreover, the second stage of the ocr pipeline/algorithm consists of three main components: feature extraction, classification, and post-processing. In the feature extraction step, unique features are extracted from each character segment to aid in later classification. The classification step maps the segmented images to their respective character classes. Finally, post-processing is performed to refine the output and ensure the accuracy of the recognized characters.

### 2.3.1 Image Pre-processing Pipeline

#### Image Acquisition

Image acquisition (IA) is the process of retrieving an image from an external source for further processing, and marks the initial stage of the OCR workflow [27]. Additionally, the process is an integral part of the digitization process that involves converting an analog image into a digital format. This retrieval process can be done via hardware such as a scanner, camera or other devices [28]. A scanner captures the content of documents and translates it into digital information represented in binary format. This stage of the OCR workflow is essential as it ensures that the captured image is of high quality, and recognizes all relevant information. Any errors or missing information in the image can negatively affect accuracy and reliability, thereby impeding the OCR softwares ability to accurately recognize the text and other important information.

In conclusion, the significance of image acquisition and ensuring it is correctly executed is fundamental in regards of achieving accurate and reliable OCR results, which are critical for applications such as digitizing older documents or automating data entry.

## Preprocessing

OCR software typically perform some basic preprocessing, as part of the OCR process. The degree of preprocessing done can vary depending on the specific engine and the complexity of the image being scanned. Imperfections are inherent to any scanner for capturing image, and noise can be present in the scanned image due to extraneous details. This noise can negatively impact the accuracy of OCR [29]. Built-in techniques used in OCR systems can improve the accuracy of the recognition process, and thus tackle the mentioned issue. Among the most basic yet crucial techniques are binarization, skew correction, noise reduction, and contrast enhancement [30]. Each technique serves a specific purpose in the preprocessing phase, and a combination can greatly improve the overall accuracy.

Similar to the aforementioned information on image acquisition, the preprocessing stage plays a critical role in enhancing the accuracy of the character recognition process. The fundamental objective of this stage is achieved by manipulating image data by eliminating noise and artifacts and rectifying any distortions or other image deformities. The captured image, from the IA phase, is typically first converted to a grayscale image, and then binarized achieving a clear demarcation between the foreground (text) and background. This process, known as binarization, is part of the digitization process and involves setting a threshold value to differentiate between foreground and background, as it is imperative for optimal character recognition by OCR systems. Thresholding is a commonly used technique, in context of image processing, to select or isolate regions of interest in an image while disregarding irrelevant parts, such as the background and image deformities [31].

In Figure 2.11, the original image appears distorted with clearly visible square boxes, referred to as pixels. Pixels are represented by numerical values, known as pixel values, that denote their intensity. For grayscale images, pixel values range from 0 to 255, where the lower values represent darker shades, and higher values represent lighter or white shades. By fixing a threshold, normally at half of the pixel range, all pixel values exceeding the threshold are considered white pixels (background), while those below are classified as black pixels.



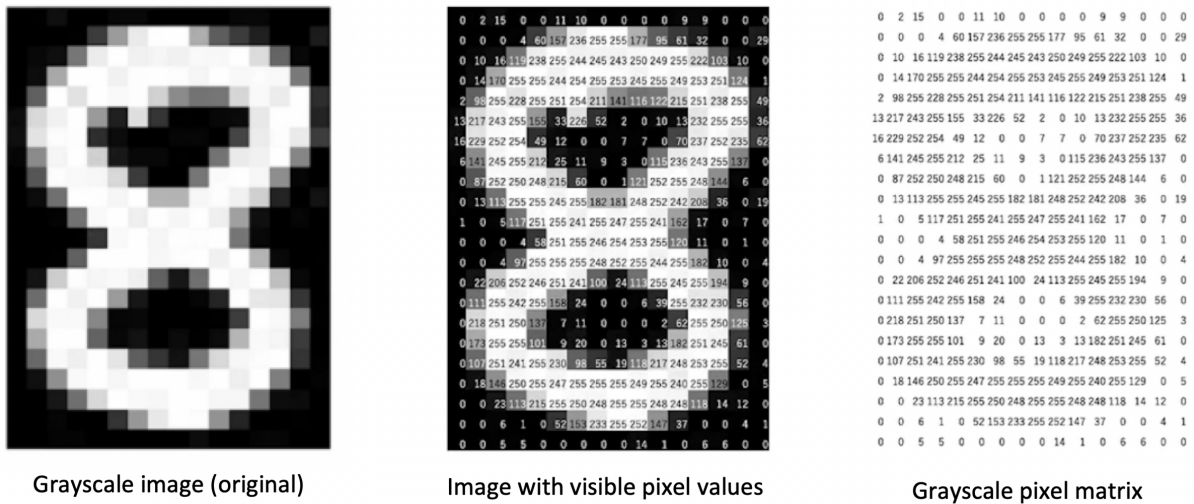


Figure 2.11: Grayscale image showcasing pixel intensity and their respective values. Source: [32]

In Figure 2.12, the conversion process during binarization is depicted, illustrating a binary image matrix. This type of matrix comprises of pixels represented by a single bit of information, i.e., either 0 or 1, making it a 1-bit image. These pixels are arranged creating a grid pattern to form the binary image matrix. The conversion process simplifies the image data, and can be achieved by either applying binarization or utilizing other suitable techniques to produce a denoised image [29]. This enables the OCR software to differentiate characters from the background with ease [33], [34].

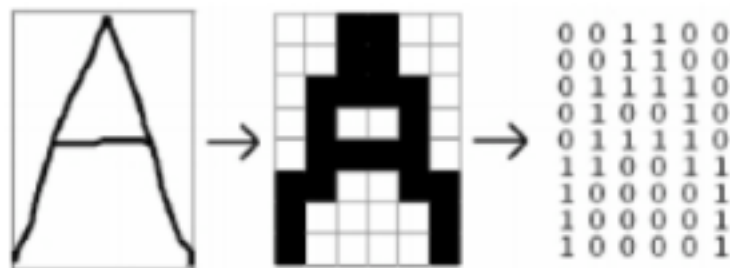


Figure 2.12: Illustration of binarization process. Original image (left), binary image (middle), and binary matrix (right). Source: Reference [35]

As previously noted, binarization is a fundamental technique that is typically integrated within OCR engines, and is performed during the initial image acquisition stage of the OCR workflow. Similarly, the same applies to other fundamental techniques mentioned earlier, and as shown in Figure 2.14 and 2.13 the techniques clearly improve image quality.

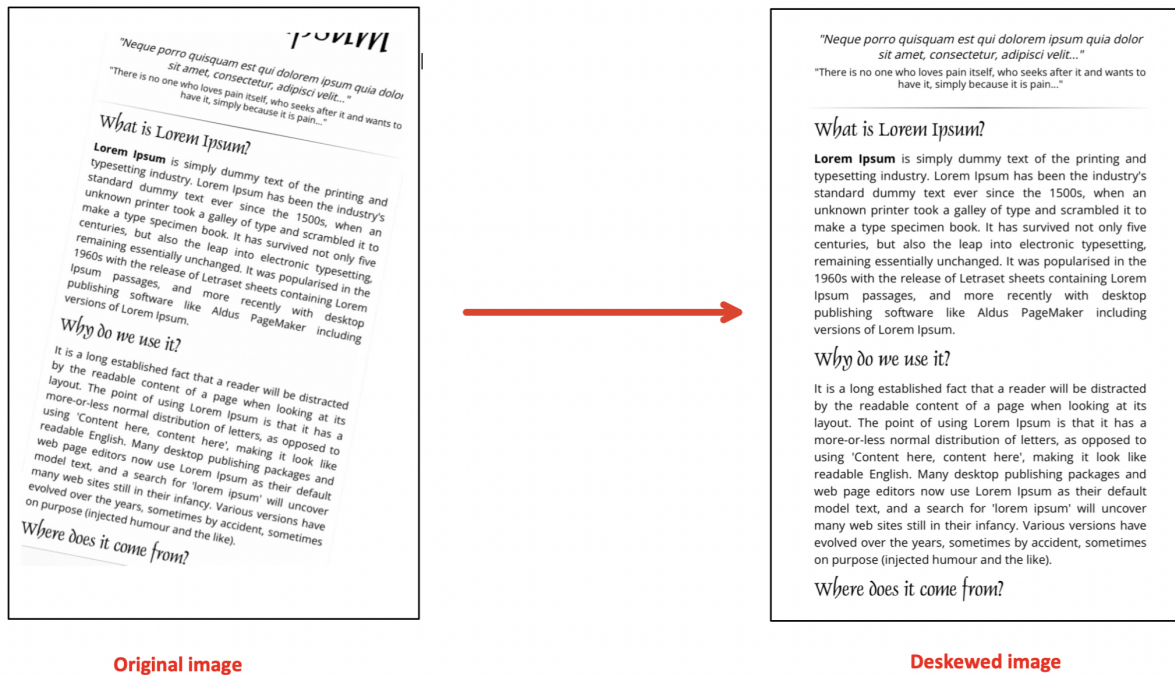


Figure 2.13: Image before and after implementation of skew correction.

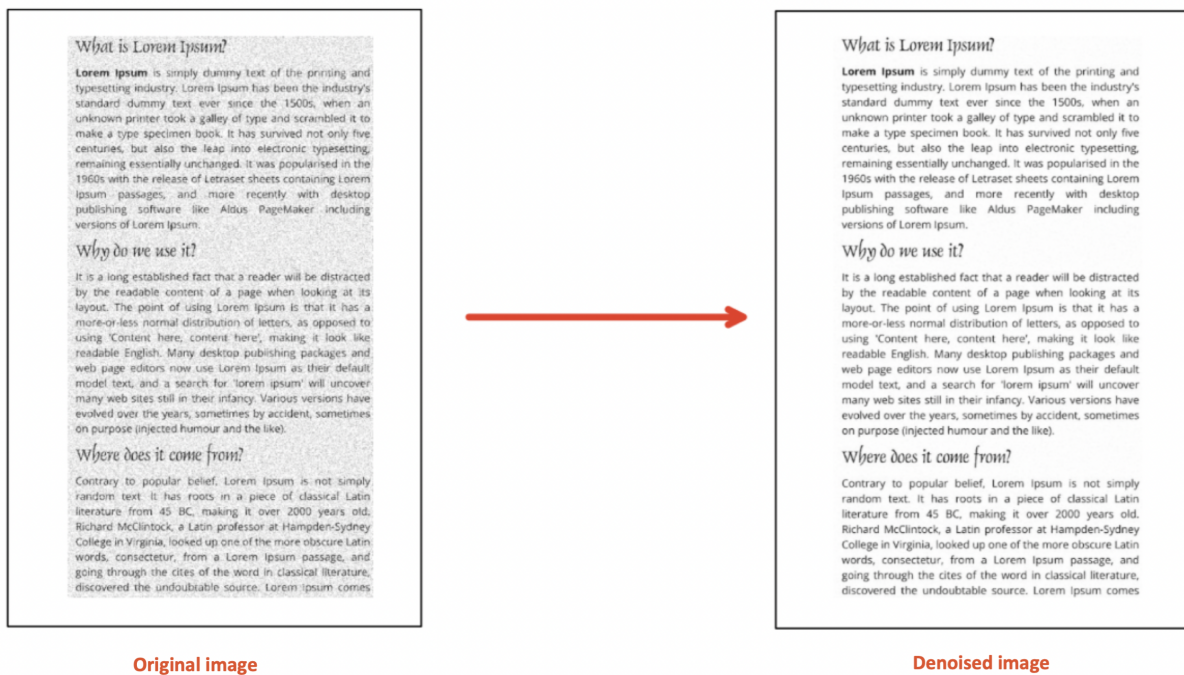


Figure 2.14: Image before and after noise reduction.

Selection of optimal processing techniques or algorithms depends on specific situations, project requirements, and the complexity of the captured image from the IA stage [36]. Moreover, the objective of this phase is to preprocess the input image or document so that it can be effectively and accurately processed by the OCR algorithm. This is accomplished by eliminating any noise and artifacts present through the use of either default techniques

provided by OCR engines or by implementing techniques that can further enhance the quality of the preprocessed image [37].

## Segmentation

Segmentation is one of the most important stages in OCR, which refers to the process of partitioning a digital image into multiple segments or subregions [38]. This allows for the extraction of individual components to be recognized and converted into machine-encoded text. The segmentation process is typically performed in a sequence of three steps, also referred to as tri-level segmentation: line level segmentation, word level segmentation, and character level segmentation. The histogram projection method is used to explain the mentioned tri-level segmentation. This method involves projecting the pixel values onto a one-dimensional axis, commonly along the horizontal or vertical axis, to generate a histogram that represents the distribution of pixels along that axis [39], [40]. The resulting histogram is also as a projection profile.

Line level segmentation is the first stage of the tri-level segmentation process, which involves detection and extraction of individual text lines from a skew corrected binary image provided from the earlier preprocessing phase. Depending on the type of OCR system (detailed explanation in Section 2.3.4 and 2.3.5), horizontal projection profile analysis is typically utilized during this level of segmentation. An example of this technique is presented in Figure 2.15. A projection profile is created by summing up the foreground pixels along each row of the image, and the resulting curve shows the distribution of black pixels along each row and is analyzed to identify the regions with high concentration of black pixels, which correspond to lines of text. The image is then segmented into smaller regions and passed through to the next level of the tri-level segmentation.

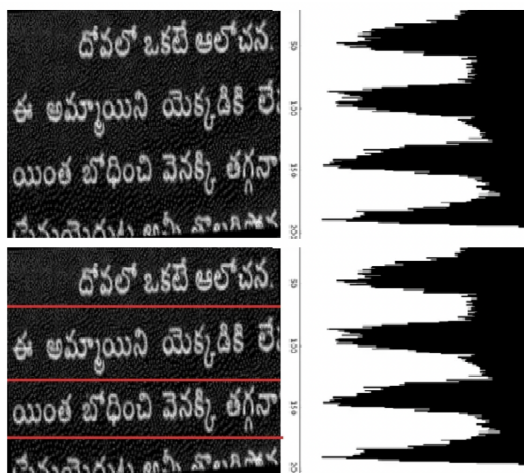


Figure 2.15: Horizontal projection profile for image segmentation. Higher peaks represent the text in a line, having a high number of foreground pixels. Rows that represent the gaps in-between the lines is represented with lower peaks. Source: [41]

The second stage of the tri-level segmentation is word level segmentation (Figure 2.17), which involves individual identification of words within a line. It is the process of determining the word boundaries in a sentence or line by computer algorithms [42]. At this level, images from the previous segmentation stage, consisting of a single line of a sequence of words, are provided. The method utilized is analogous to the previous, with the only difference being that the OCR performs vertical projections on the image rather than horizontal projections [43]. By vertically projecting the binary subregions provided, a projection profile is generated, which is illustrate in Figure 2.16.

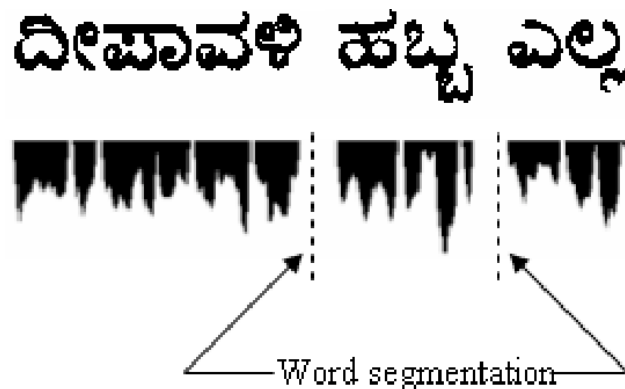


Figure 2.16: Vertical histogram projection of a line of text. Higher peaks indicate high number of foreground pixels. Gaps in-between words is represented by lower peaks indicating high number of background pixels. Source: [44]

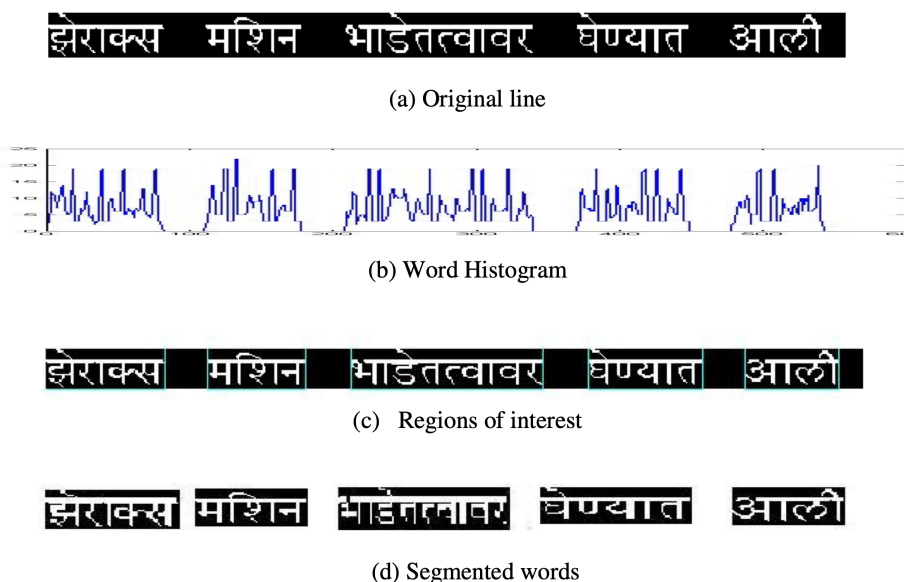


Figure 2.17: (a) Image provided from the line segmentation stage. (b) Vertical projection profile for the provided image. (c) Detected word boundaries after analyzing the projection profile. (d) Final segmented words. Source: [43]

After partitioning the images provided from the previous segmentation level, the segmented words are passed through to the final level of the tri-level segmentation process

known as character level segmentation. Similarly to the previous level, the OCR performs vertical projections on the images containing a single word consisting of a sequence of characters [45]. Additionally, OCR systems use several techniques to accurately extract and recognize each character, such as thresholding [46].

Depending upon the context where the OCR is being used, this level of segmentation can be more or less complicated. If the OCR system is being applied to text, where the characters within a word are separate, maintaining a relatively uniform gap in-between, character level segmentation may be less complicated and the OCR can segment the characters by leveraging the small gaps between characters. On the other hand, if the characters within a word are joined, over-segmentation may occur resulting in a fragmented representation that may not be recognizable as text.

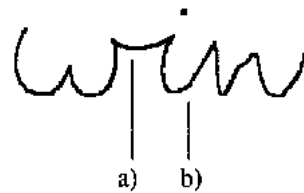


Figure 2.18: Example of ligature in cursive handwriting. (a) An extra stroke. (b) A smooth connection. Source: [47]

The concept of ligatures in cursive handwriting can lead to over-segmentation problem, which is when the OCR system fails to distinguish the ligature with the curve from open characters [48], and rather considers both as valid candidates for segmenting. Hence, splitting the image into more segments than necessary, which can lead to errors in recognition. Various techniques exist to address this issue, including clustering and skeletonization.



Figure 2.19: Word images showing all types of segmentation errors. Source: [49]

Segmentation plays a pivot role in OCR, as it encompasses the partitioning of the input image into smaller subregions such as lines, words, and characters, which can then be recognized and translated into machine-encoded text. Line level segmentation identifies

individual lines of text, word level segmentation detects individual words, and character level segmentation extracts individual characters. The sequential tri-level segmentation process enables the OCR system to accurately recognize and translate the input image into digital text.

### 2.3.2 Text Recognition Pipeline

OCR systems play an important role in regards of automated recognition and interpretation of printed or handwritten text from images. The text recognition pipeline is an essential component of these systems, consisting of three main steps: feature extraction, classification, and post-processing.

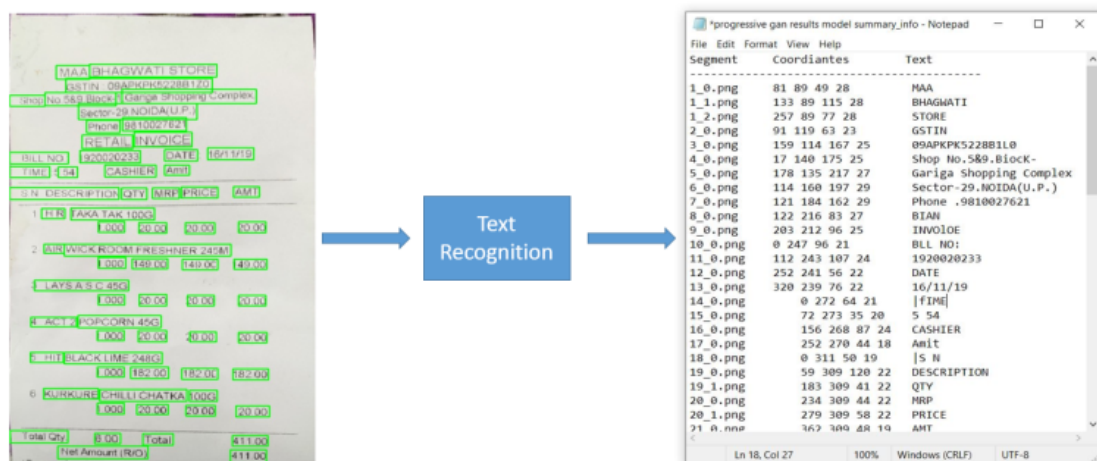


Figure 2.20: Simple example of text output (right) after text recognition. Source: [50]

### Feature extraction

Feature extraction involves extracting relevant information from the input image to identify individual characters or text. Its main objective is to obtain the most relevant information contained from the original data, reducing the complexity of the input data while retaining relevant information [51]. Features such as lines, curves, or texture patterns, can be extracted through application of techniques such as edge detection, blob detection, or template matching. Furthermore, these features are transformed into a numerical representation that can be used as input to machine learning algorithms for classification. Overall, feature extraction aims to transform the input data into a format that can be processed and analyzed by machine learning algorithms, allowing for accurate and efficient recognition of text.

## Classification

Classification is the process of using machine learning algorithms to classify the extracted features, from the earlier step, as specific characters or words. It is the procedure of distributing inputs, with respect to detected information, to their comparing class in order to create groups with homogeneous qualities, while segregating different inputs into different classes [52]. Typically, this involves training a classifier using a labeled dataset to learn to recognize different letters, digits, and symbols.

## Post-processing

Post-processing involves refining the results from the classification step to improve the overall accuracy of the OCR system. This may be achieved through the application of various methods, and the least complex for consolidating the context data is the usage of a dictionary for amending minor errors of the OCR. The underlying thought of the post-processing step is to perform spell-checking on the OCR output and provide various options for the recognized text using a dictionary [52].

The text recognition pipeline is a critical component in OCR systems, enabling the automated and accurate recognition of text from images. Understanding the underlying mechanisms and optimizing the pipeline for specific applications can highly improve the OCR performance, and benefiting a wide range of industries.

### 2.3.3 Long Short-Term Memory

Long short-term memory (LSTM) is a subtype of recurrent neural network (RNN) that is commonly utilized for sequence modeling and processing [53]. Recurrent neural networks are designed to leverage artificial memory processes that can aid these artificial intelligence programs to more effectively emulate human thought [54].

LSTMs are a key component in OCR systems and is usually used in combination with other techniques such as Convolutional Neural Networks (CNNs) and Attention Mechanisms, to perform tasks such as character recognition, word segmentation, and text recognition [55], [56]. Typically, OCR systems are encountered by variable-length input sequences, and LSTMs can learn to model the temporal dependencies in sequential data, making them suitable for handling such inputs.

### 2.3.4 Types of Optical Character Recognition Systems

Based on the type of input, OCR systems are divided into two main categories [57]: handwriting recognition and machine printed character recognition. Machine printed character recognition is a simpler problem because characters are usually of uniform dimensions, and the positions of characters on the page can be predicted. In contrast, handwritten recognition is a much tougher job due to different writing styles of users as well as different pen movements by the user for the same character. OCR systems can also be categorized based on image acquisition mode, character connectivity, font-restrictions, and other criteria.

### 2.3.5 OCR Engine

OCR systems consist of various components such as image pre-processing, OCR engine, post-processing, and UI. The OCR engine is the core component of an OCR system, responsible for converting image data into text. It is a software library or module that provides character recognition functionality to the OCR system.

#### Tesseract OCR

Tesseract OCR is an open-source OCR engine [58], that is primarily designed to recognize machine-printed characters in various documents like books and newspapers. Depending on the application's requirements, this OCR engine can be integrated as both an offline and online OCR system. As an offline system, the engine processes static data, i.e., images that have already been captured, whereas the online mode requires additional software and hardware components to capture and process input in real-time. However, Tesseract OCR is better suited for offline applications, where the input data can be preprocessed and optimized for recognition before being fed into the engine.

### 2.3.6 Configurations

OCR configuration refers to the various customizable settings and options within an OCR system, which can be fine-tuned to optimize the recognition process for specific use cases or applications. Provided below are general configuration settings that can be utilized.

- Character recognition settings: These can include options for adjusting the level of detail used to recognize individual characters, such as adjusting the font size or font type recognition thresholds.



- Character set and dictionary: Can be configured to recognize or exclude specific characters or symbols.
- Output format: Can be configured to output text in various formats, such as plain text or PDF, depending on the desired application and use cases.
- Language support: OCR engines can be configured to recognize specific languages or character sets.
- Page segmentation mode: Determines how the OCR engine identifies and separates text and other elements on a page, such as tables or images.
- OCR engine mode: The setting determines the overall approach used by the OCR engine to recognize characters, such as using a neural network-based approach.

Customization of OCR systems is an important aspect in achieving optimal results. The available configurations in an OCR system are significantly influenced by the specific algorithms and approaches employed by the OCR engines. As discussed in Section 2.3.5, the engine is the central component of an OCR system, and the degree of customization can therefore vary depending on the particular engine used.

# Chapter 3

## Suggested Approach

### 3.1 Design

The graphical user interface (GUI) is a critical component of the hardware documentation process. It comprises various widgets or elements that serve a specific purpose in this application. Most of the GUI is covered by the scene widget, which displays the current image the user is working on. Below the scene widget, there are three widgets, two containing a list each, and the last widget containing all the buttons for the workflow. The widget containing the buttons, also contains information presented to the user during certain stages of the workflow.

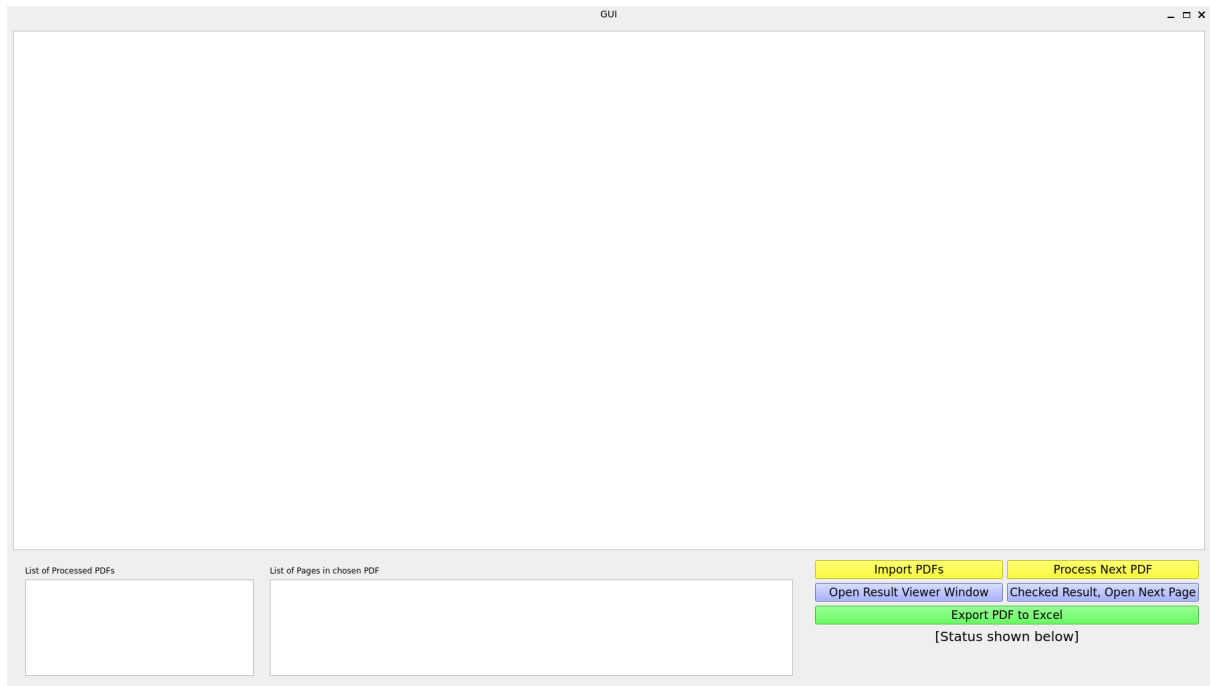


Figure 3.1: Screenshot of the interface's main window layout.

The button widgets are conveniently arranged to follow a natural flow that guides the user through the workflow of the GUI. The buttons take you through the workflow of the GUI, where you start with the uppermost left button, followed by the uppermost right button. By slowly working your way down the buttons, you reach the final button which is color-coded green indicating the completion of the workflow. Each "line of buttons" corresponds to its "line of work", with the top layer of buttons handling importing and processing a PDF, the second layer handling information extraction and display, and the last layer exporting the corrected information. The "line of work" provides an indication of what belongs together by matching the buttons' colors, each "line of work" then represented by the same color. Functionally, the buttons are rendered unusable until specific criteria in the workflow have been achieved, which is explained in Section 3.2. This helps in preventing the user from accidentally performing an action prematurely.

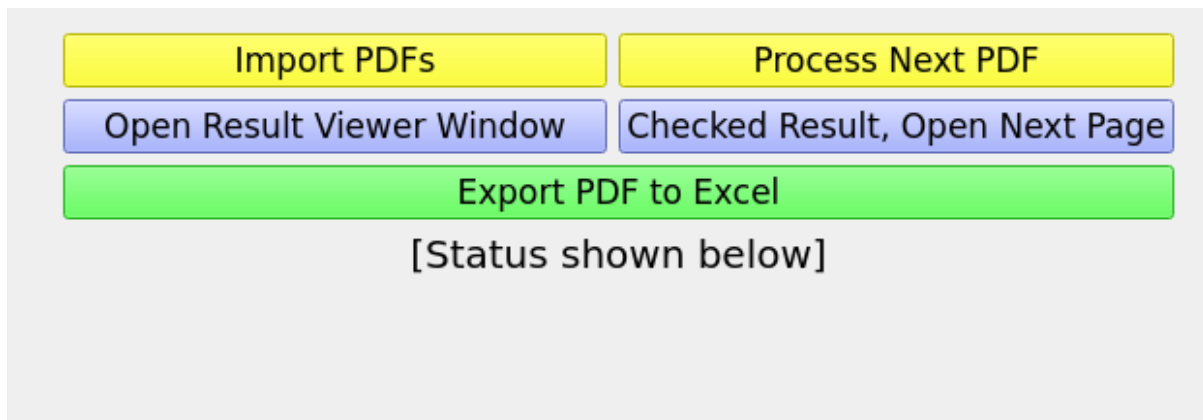


Figure 3.2: Screenshot of the button widget's layout.

The Result Viewer Window displays the extracted information to the user, and is provided in Figure 3.3. There are two buttons at the top of the Result Viewer Window for adding or removing a loop at the bottom. Each loop spans most of the window and is presented as an element in a big list. Each loop has 13 sections corresponding to the 13 words to be extracted. Each section displays the element to be found at the top, followed by the corresponding word the software has recognized from the displayed image. A list of all other potential matches appears under the recognized word. Each element in the list is clickable, leading to the recognized word being updated. If the element is not in the list, the user can manually type the correct word in the input field, which overwrites the value in the found word. The input field has a placeholder text "Enter Word" if the user has added a loop manually using the "Add Loop" button.

The GUI is designed to be used with two screens, with the Result Viewer Window appearing in a separate window, allowing the user to review the words on one screen while simultaneously matching the result from the image of the loops in the main window GUI. To accommodate for the loop's width in an image, the image viewer in the GUI layout

takes up the majority of the main window. The user can zoom in and out of the picture and drag the image around the viewer, giving a seamless user experience.

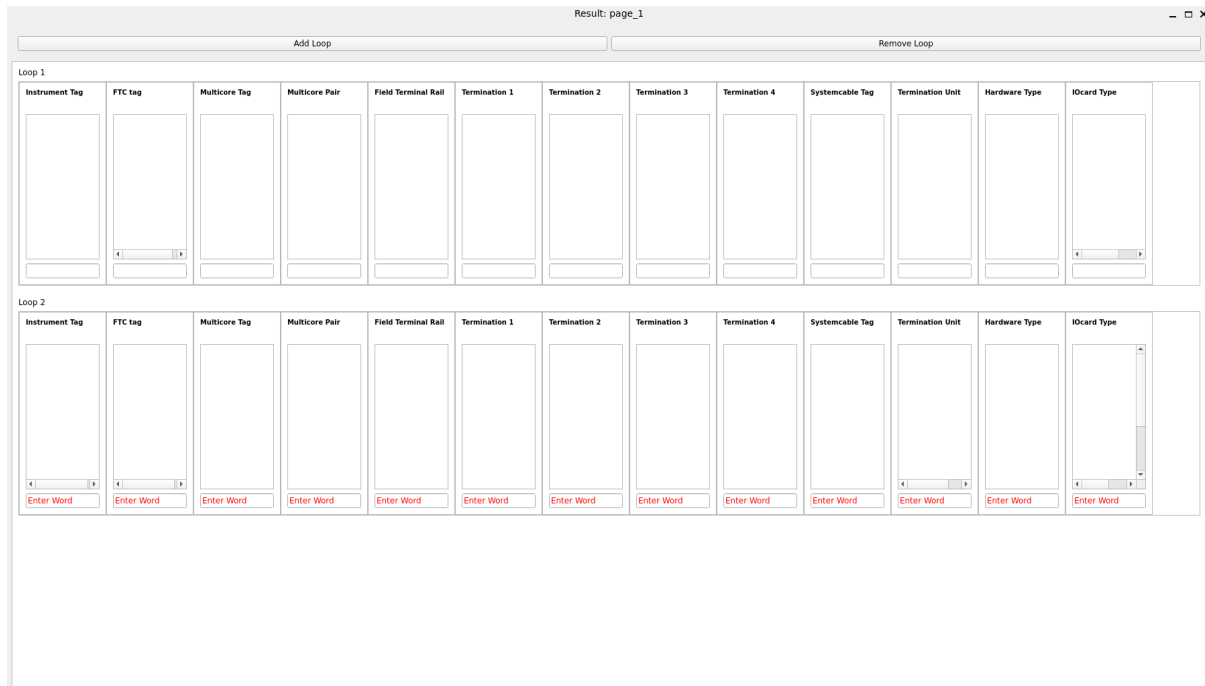


Figure 3.3: Screenshot of the Result Viewer Window

Once the user completes the validation and correction of what the software has extracted, the next image can be opened using the "Checked Result, Open Next Page" button. A green-colored icon marks the selected image from the list of images to indicate that the user has validated the current image. A label above the image list widget is updated to provide this information. The next image is selected from the list of images and displayed in the image viewer, and the Result Viewer Window is updated with information from the OCR file for the new image. If necessary, the user can redo the previous image by selecting the image from the image list and clicking the "Open Result Viewer Window" button.

Overall, the GUI design provides a seamless and intuitive experience for the user, facilitating efficient and accurate hardware documentation process.

## 3.2 Implementation

### 3.2.1 Import PDFs and Begin Processing

The GUI features two buttons for PDF processing. The first button, labeled "Import PDFs", is located at the top-left corner of the button widget. When clicked, this button

invokes the `importPdfDlg` method of the parent class, which displays a popup dialog prompting the user to import one or several PDFs for further processing. As shown in Figure 3.4, the user can select a single PDF or multiple PDFs. The `importPdfDlg` method retrieves a list of file paths from the dialog box and copies them from the source to an internal "input" folder, which the GUI uses to access the PDF files. This functionality simplifies the user experience by eliminating the need for the user to locate the program's file directories manually.

```
1     def importPdfDlg(self):
2         """Use the Qt file open dialog to select one or several PDFs to
3         import."""
4         options = QFileDialog.Options()
5         options |= QFileDialog.DontUseNativeDialog      # make dialog
6         appear the same on all systems
7         flt = "All files (*)"
8         (fName_list, used_filter) = QFileDialog.getOpenFileNames(parent
9         =self, caption="Import PDFs",
10        filter=flt, options=options)
11
12        for i in fName_list:
13            shutil.copy(i, os.path.join(BASEDIR, INPUT, os.path.split(i)
14            )[1]))
15
16        self.buttonWindow.info_label.setText(f"Imported selected PDFs
17        to unprocessed folder")
18        return
```

Listing 3.1: Code snippet of the `importPdfDlg` method.

The "Import PDFs" button does not constrain the workflow of the program. Additional PDF files can be imported for processing at any time by the user.

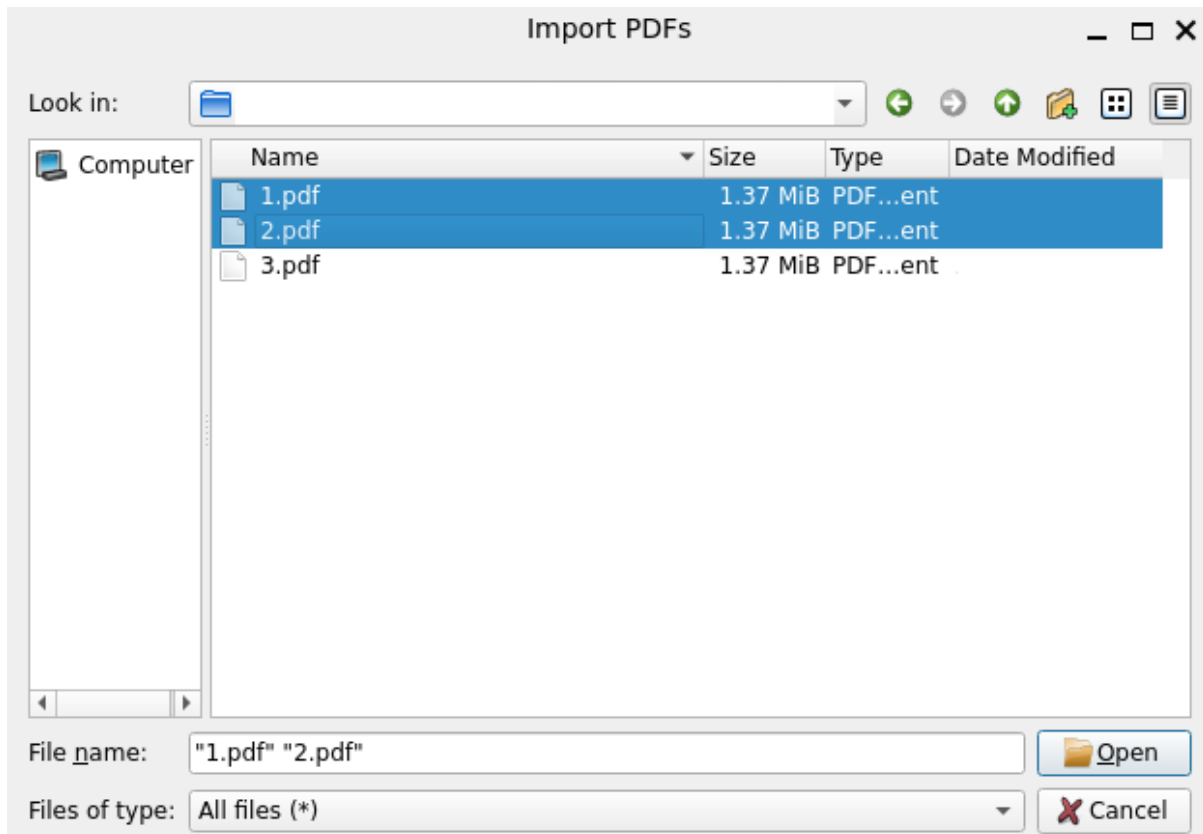


Figure 3.4: Import PDF dialog box

The second button, labeled "Process Next PDF," is located at the top-right corner of the button widget. When clicked, this button invokes a function in another thread, explained in Section 3.2.2, that starts the primary processing steps of the PDFs. This includes converting them to images, which the GUI processes using OpenCV and enabling OCR.

The "Process Next PDF" button remains inactive until certain criteria are satisfied. Two conditions must be met. The first criterion requires the input processing folder to contain at least one PDF file, to ensure the function's execution. The second criterion is more lenient; if an error is detected, the function automatically corrects the error, thereby allowing it to resume processing. For an error to occur, an internal folder structure with the PDF's name must be absent. The code snippet in 3.2 details these checks below.

```

1  def processNextPDF(self, consecutive=False):
2      """Get the next PDF in the directory"""
3      try:
4          root, pdf_name_end = next(self.PDF_generator)
5          pdf_name = pdf_name_end.split(".")[0]
6      except StopIteration:
7          pdf_name = ""
8
9      if pdf_name == "":

```

```

10         self.buttonWindow.info_label.setText(f"No more PDFs in
unprocessed folder, please import first")
11         return
12         if not os.path.exists(os.path.join(BASEDIR, OUTPUT, pdf_name,
PDF_PAGES)):
13             os.makedirs(os.path.join(BASEDIR, OUTPUT, pdf_name,
PDF_PAGES))

```

Listing 3.2: Code snippet of the processNextPDF method.

If the 'self.PDF\_generator' variable, does not return a PDF path, the function terminates prematurely, and the user is notified of the error. It is worth mentioning that this variable executes the code from Listing 3.1 and 3.2.

### 3.2.2 Converting PDF

This section addresses the computational requirements for the conversion of PDFs to images and the extraction of OCR data. To achieve this, a worker class, known as the 'Converter Worker', was created to handle the conversion process. This worker is moved to another thread, creating ideal conditions for running a heavy computational task in the background while not interfering with the GUI in the main thread. Additionally, the use of threads enabled the worker to emit status updates that the main thread could listen for, thereby keeping the user informed of the background task's progress. A code snippet of the 'Converter Worker' class is presented in Listing 3.3.

```

1 class ConverterWorker(QObject):
2     finished = Signal()
3     pdf_finished = Signal()
4     pdf_label_worker = Signal(str)
5
6     def __init__(self, parent=None):
7         super(ConverterWorker, self).__init__()
8         self.pdf_path = None
9         self.output_folder = None
10        self.pdf_name = None
11
12    def start(self):
13        self.pdf = pdfium.PdfDocument(self.pdf_path)
14        n_pages = len(self.pdf)
15        for page_number in range(n_pages):
16            QApplication.processEvents()
17            self.pdf_label_worker.emit(f"Currently converting PDF to
images...{page_number}")
18            self.process_pdf(page_number)
19            os.remove(self.pdf_path)

```

```

20     self.finished.emit()
21     self.pdf_finished.emit()
22
23     def process_pdf(self, page_number):
24         page = self.pdf.get_page(page_number)
25         pil_image = page.render_topil(
26             scale=5,
27             rotation=0,
28             crop=(0, 0, 0, 0),
29             greyscale=False,
30             optimise_mode=pdfium.OptimiseMode.NONE,
31         )
32         pil_image.save(os.path.join(self.output_folder, f"page_{
page_number+1}.png"))

```

Listing 3.3: Code snippet of the ConverterWorker class.

Before the 'Converter Worker' begins its task, the worker's attributes are initialized in the init method. Once called by the calling worker, the start method of the 'Converter Worker' is invoked, which loads the PDF into a variable and iterates over its pages. For each page, the 'process\_pdf' method is executed, and a signal is emitted to the upper worker to provide the user with status information regarding the current page being processed.

The signal must be outside the scope of the function performing the heavy lifting. Otherwise, the event loop ends up being blocked by the long-running operation, rendering the program unresponsive. The label in the GUI gets bombarded as soon as all pages have been converted.

By employing the use of threads, the program remains interactive while performing heavy computation in the background. The 'Converter Worker' class plays a crucial role in the PDF to image conversion process. Its 'process\_pdf' method renders the given PDF page, scales it, and converts it to an image, using a scaling factor of 5, which provides the highest number of pixels per inch without exceeding the function's threshold. Upon conversion, the newly converted image is stored locally on disk for the subsequent worker to access. The worker emits a finished signal once the last page has been converted to an image, prompting the image processing worker to start its task.

### 3.2.3 Image Processor

```

1 class ImageProcessorWorker(QObject):
2     finished = Signal()
3     ocr_dict_worker = Signal(tuple)
4     page_label_worker = Signal(str)

```



```

5
6     def __init__(self, parent=None):
7         super(ImageProcessorWorker, self).__init__()
8         self.directoryPath = None
9         self.image_names = None
10        self.curWorkingFileNamePage = None
11
12    def start(self):
13        for i, picture in enumerate(self.image_names):
14            self.page_label_worker.emit(f"Currently performing OCR on {
15i+1}/{len(self.image_names)}: {picture}")
16            self.process_image(picture)
17            self.page_label_worker.emit(f"Finished OCR on all pages")
18            self.finished.emit()
19
20    def process_image(self, picture):
21        image = cv2.imread(os.path.join(self.directoryPath, self.
22curWorkingFileNamePage, PDF_PAGES, picture))
23        image = self.addFilterImage(image)
24        self.ocr(image, picture.split(".")[0])
25
26    def addFilterImage(self, image):
27        image = removeLinesAndBorders(image)
28        image = removeNoise(image)
29        return image
30
31    def ocr(self, ocr_compatible_image, picture_name):
32        custom_config = r'--oem 3 --psm 11' # OCR engine configuration
33        ocrDict = pytesseract.image_to_data(ocr_compatible_image,
34config=custom_config, output_type=Output.DICT)
35        self.ocr_dict_worker.emit((picture_name, self.
36curWorkingFileNamePage, ocrDict))

```

Listing 3.4: Code snippet of the ImageProcessorWorker class.

The Image Processor Worker plays a crucial role in the PDF to text conversion process by processing the images extracted from the PDFs and extracting text using OCR. The worker is initiated by the PDF worker and emits signals to update the status label in the GUI, send the extracted text to the main thread, and indicate when it has completed processing all the images.

The start method of the Image Processor Worker operates on a folder containing images corresponding to the PDF being processed. For each image, the worker loads the image into memory and applies filters to remove noise and contour, thus performing image preprocessing.

After the image has been loaded into memory, the addFilterImage method is used.

This method calls further functions for processing the in-memory image. The image is cautiously converted to a grayscale image before line and border detection begins. All the contours of the image are obtained. New contour lines in the color white are drawn on top of the contour lines in black, essentially removing the original contour lines. After this step, simple background noise in the image is removed.

The preprocessed image is now ready for OCR. The ocr method employs a Python wrapper called pytesseract to interact with the Tesseract interface. The Tesseract response contains all the raw data found, which is stored in a dictionary. The dictionary contains keys for the word hierarchy, coordinates, and confidence levels. The Tesseract configuration was set to `--oem 3 --psm 11`, with OEM (OCR Engine Mode) in "Legacy + LSTM engines" mode and PSM (page segmentation mode) in "Sparse text. Find as much text as possible in no particular order" mode, which produced the most reliable output. After analyzing the OCR output's confidence level and the completeness of the extracted information, we found that this particular set of configuration parameters produced the most accurate output. Other page segmentation modes were unsuitable for the sparse text in this specific use case. Figure 3.5 shows the difference by implementing the specified configuration parameters.

5; 1; 8; 1; 1; 4; 7555; 277; 286; 81; 1; 13-IF	No configuration
5; 1; 8; 1; 1; 5; 7868; 273; 328; 115; 83; -00118	
5; 1; 8; 1; 1; 4; 7555; 277; 674; 93; 87; 13-DF-00118	Configuration parameters: <code>--oem 3 --psm 11</code>

Figure 3.5: The text on the right of the semicolon is the recognized text, and its respective confidence level (as a percentage) at the left side of the semicolon.

The Tesseract output is then sent to a function in the main thread as a tuple, and the heavy processing is done. The dictionary keys output by Tesseract are `level`, `'page_num'`, `'block_num'`, `'par_num'`, `'line_num'`, `'word_num'`, `left`, `top`, `width`, `height`, `conf`, and `text`. `Level` indicates the current found object's sorting into `'page_num'`, `'block_num'`, `'par_num'`, `'line_num'`, or `'word_num'`. The `left`, `top`, `width`, `height` keys give the coordinates of the found word. The `conf` key indicates the Tesseract engine's confidence in its accuracy and the word found. The `text` key gives the word found. This hierarchy has minimal use for the tool, with the exception of `word_num`, which can be used to find the word following the word being searched for.

Each key in the Tesseract dictionary has a list where the index corresponds to matching information. For instance, the value in index 39 of key `level` corresponds to the value in index 39 of key `'page_num'`. The function in the main thread stores all the related data, one of each key in the given index, as a single line. The information is then checked against a replacement dictionary and stored to disk. The replacement dictionary replaces

inaccurate Tesseract readings with predefined dictionary words, improving the accuracy of extracted words.

The function handling the Tesseract output can operate in the main thread without risking rendering the GUI unusable. Each key in the Tesseract dictionary has a corresponding list, with the index in that list indicating matching information. The function in the main thread consolidates all the related data - one of each key in the given index - into a single line. Each line is checked against a replacement dictionary and then stored on disk.

Once all the images have undergone OCR extraction, a signal is emitted indicating that the worker has finished processing. The status field is also updated to reflect this. By the time the PDF is entirely processed, an item reflecting the imported PDF's name appears in the list of processed PDFs in the main GUI. Upon clicking the item, the second list in the GUI populates with items corresponding to all the processed pages from the clicked PDF.

The use of the Image Processor Worker enables efficient extraction of text from images, with the worker being initiated by the PDF worker and emitting signals to update the status label in the GUI, send the extracted text to the main thread, and indicate when it has finished processing all the images. The ocr method utilizes the pytesseract wrapper to interact with the Tesseract OCR engine, and the function handling the Tesseract output can operate in the main thread without risking rendering the GUI unusable.

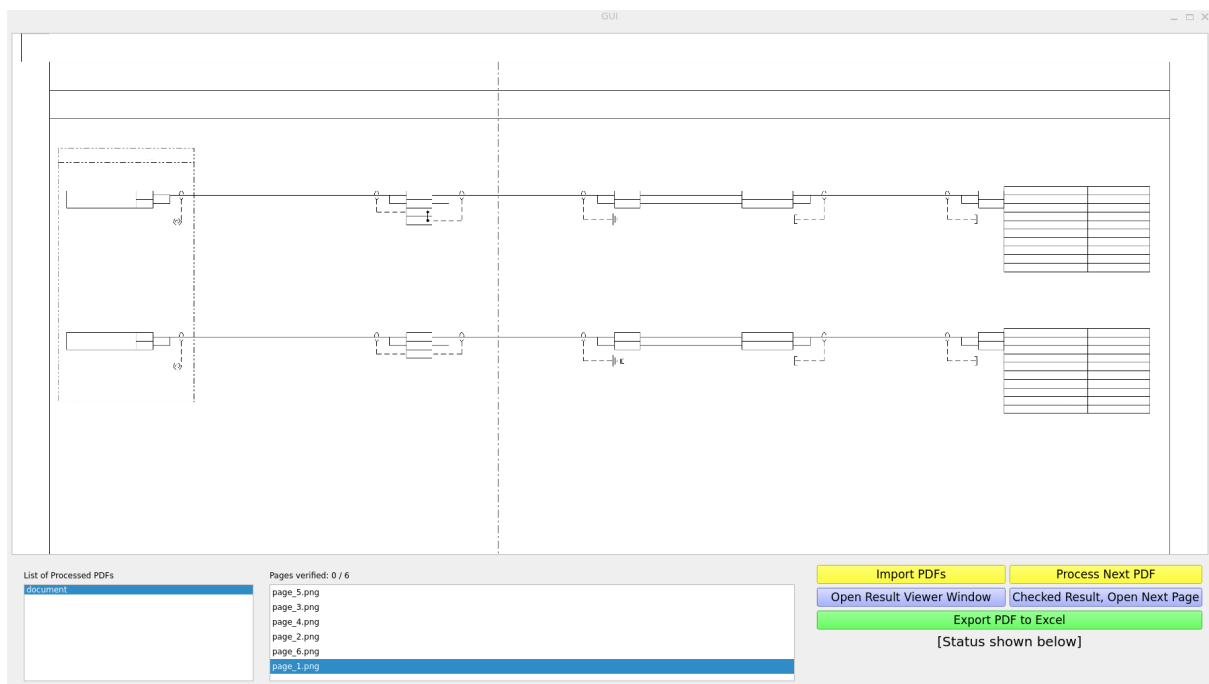


Figure 3.6: Main Window after processing one PDF, and file "page\_1.png" being open.

### 3.2.4 Hardware Extraction

This section presents a comprehensive program analysis of the OCR output, a critical element of the hardware information extraction process. Prior to presenting results to the user, an instance of the "searcher" class is created, which takes the OCR output file as a parameter. This class has an attribute for every word it is tasked with finding, and each attribute is responsible for finding its specified word using either regex or a search criteria.

The searching of words is mostly based on a baseline, which is created by finding every occurrence of the word associated with an instrument tag. The instrument tag is found using regex. The baseline is then given as the coordinates of the different instrument tags found. Every loop is based on the instrument tag, and the other words are to be filled in. For each word following the instrument tag, a search is conducted based on the baseline. This way, we can differentiate between the same words that belong to different loops.

```

1 class hardware_information():
2     def __init__(self, ocr, master=True):
3         self._filepath = ocr
4
5         self.initiate_read()
6
7         if master:
8             self.instrumentTag = instrumentTag(self._file_word_list, "
Instrument Tag")
9             self.get_baseline()
10            self.generate_hardware_information(master)
11
12    def generate_hardware_information(self, master=False):
13        words_exclude = ["i/o", "card", "type", "vo", "system", "
termination", "unit", "hardware", "typ", "typ.", "h/w", "i/o"]
14        if not master:
15            self.instrumentTag = instrumentTag(self._file_word_list, "
Instrument Tag", self._baseline)
16            self.ftcTag = ftcTag(self._file_word_list, "FTC tag", self.
_baseline)
17            self.multiCoreTag = multiCoreTag(self.ftcTag.section, "
Multicore Tag")
18            self.multiCorePair = multiCorePair(self.ftcTag.section, "
Multicore Pair", self.ftcTag.int_list)
19            self.fieldTerminalRail = fieldTerminalRail(self.ftcTag.section,
"Field Terminal Rail")
20            self.termination1 = termination(self.ftcTag.section, "
Termination 1", self.ftcTag.int_list)
21            self.termination2 = termination(self.ftcTag.section, "
Termination 2", self.ftcTag.int_list, self.termination1.height)

```

```

22     self.termination3 = termination(self.ftcTag.section, "
Termination 3", self.ftcTag.int_list, self.termination2.height)
23     self.termination4 = termination(self.ftcTag.section, "
Termination 4", self.ftcTag.int_list, self.termination3.height)
24     self.systemCableTag = systemCableTag(self._file_word_list, "
Systemcable Tag", self._baseline, words_exclude=words_exclude)
25     self.terminationUnit = terminationUnit(self._file_word_list, "
Termination Unit", self._baseline, words_exclude=words_exclude)
26     self.hardwareType = hardwareType(self._file_word_list, "
Hardware Type", self._baseline, words_exclude=words_exclude)
27     self.IOcardType = IOcardType(self._file_word_list, "IOcard Type
", self._baseline)
28
29
30
31     def initiate_read(self):
32         self._file_word_list = []
33         with open(self._filepath) as f:
34             for index, line in enumerate(f):
35                 elements = line.split("; ")
36                 if elements[0] != "5":
37                     continue
38                 else:
39                     self._file_word_list.append(OCRWORD(elements[11],
elements, index))
40         return

```

Listing 3.5: Code snippet of the 'hardware\_information' class.

The ftc tag, which is also found using regex, is another critical word to locate. Over half of the words to be found are derived from this tag. By using coordinates, we can derive most of the other words since they are typically grouped together and situated around the ftc tag in most of the documents. The OCR output file is processed for each word, and only the words in the given area by the ftc tag are considered potential correct words.

The SystemNode class is responsible for filtering the word list based on the baseline. This class takes in a word list, a word, a regex string, a baseline, a baseline length, a within-same flag, and a list of excluded words. The class filters the word list by the baseline and excludes any words in the excluded words list. If only one word is found, it is approved as the correct word.

```

1 class SystemNode():
2     def __init__(self, word_list, word, regexString="", baseline=0,
baseline_length=300, within_same=True, exclude_words=[]):
3         self.systemWord = word
4         self.approved_word = None

```

```

5     self.baseline = baseline
6     self.baseline_length = baseline_length
7     self.word = None # Is always appended to
8     self.word_list = word_list
9     self.regexMatches = []
10    self.word, self.possible_match, self.search_word = try_get_word
    (self.word_list, self.list_of_search, within_same)
11    if regexString != "":
12        self.regexMatches, self.word = self.regex(regexString)
13    self.word = merge_words(self.word)
14    if exclude_words != []:
15        self.word = [x for x in self.word if not any(word.lower()
in x.word.lower() for word in exclude_words)]
16    self.filter_word_list_by_baseline()
17    if len(self.word) == 1:
18        self.approved_word = self.word[0]
19    return

```

Listing 3.6: Code snippet of SystemNode class.

## Validation of Hardware Extraction

Hardware information extraction is a crucial component of the hardware documentation process. The extraction of relevant hardware information is accomplished by analyzing the OCR output file through instances of different classes that are responsible for finding specific words.

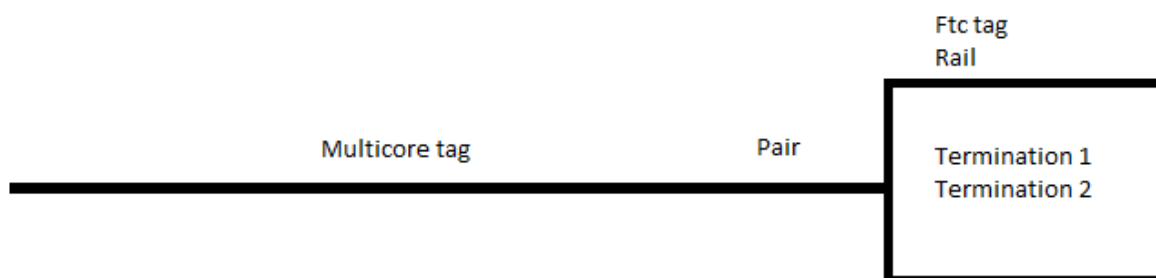


Figure 3.7: Example of typical layout of words relative to the ftc tag

The ftcTag class is responsible for finding the ftc tag using regex. A smaller search list is generated, holding all the words in proximity of the found ftc tag. The rail is found by searching for the topmost word (by coordinates) in the search list that is also right under the ftc tag. Terminations and pairs are solely numbers, and the search list is filtered by

words that can also be integers. Multicore tag is found by applying a regex on the search list while choosing the leftmost appropriate word based on coordinates.

```

1 class ftcTag(SystemNode):
2     def __init__(self, word_list, word, baseline):
3         self.list_of_search = ["ftc", "tag"]
4         regexString = re.compile(r"^[a-zA-Z]{1})-13-((?!JSE|CP|ER)[a-zA-Z]{2,5})-([0-9]{4,5})([a-zA-z]{0,1})")
5         super().__init__(word_list, word, regexString, baseline=
6         baseline)
7         self.section = try_get_all_words_under(self.approved_word, self
            .word_list)
            self.get_int_from_section()

```

Listing 3.7: Code snippet of SystemNode class.

The `saveResultOpenNext` function saves the processed loops and opens the next page. The function first verifies the existence of an active Result Window and an active Result File Name. If both conditions are met, the finished processed loops are fetched and stored in a text file associated with the current working image. The directory list is then updated to indicate that the current page has been verified. If the current page is the last page, the function returns. Otherwise, the function opens the next page and the Result Viewer Window.

```

1     def saveResultOpenNext(self):
2         if self.curResultWindow is None:
3             self.buttonWindow.info_label.setText("Please have an active
4             Result Window")
5             return
6         if self.curWorkingResultFileName is None:
7             self.buttonWindow.info_label.setText("Please have an active
8             Result Window")
9             return
10
11         with open(os.path.join(self.directoryPath, self.
12         curWorkingFileNamePage, self.curWorkingFileName + ".txt"), "w") as f
13         :
14             loop_list = self.curResultWindow.get_words_loops()
15             for loop in loop_list:
16                 f.write(f"{loop}\n")
17             curr = self.directoryWindow.directoryList.currentRow()
18             self.directoryWindow.directoryList.item(curr).setData(1, QColor
19             (0, 255, 0))
20             self.directoryWindow.directoryList.setCurrentRow(curr + 1)
21             files_processed = [i.split(".")[0] for i in os.listdir(os.path.
22             join(self.directoryPath, self.curWorkingFileNamePage)) if i.endswith
23             (".txt")]

```

```

17     self.directoryWindow.pages_label.setText(f"Pages verified: {len
    (files_processed)} / {len(self.directoryWindow.directory)}")
18     if curr == self.directoryWindow.directoryList.count() - 1:
19         return
20     self.openFile(self.directoryWindow.directoryList.item(curr + 1)
    .data(3))
21     self.openResultWindow()

```

Listing 3.8: Code snippet of saveResultOpenNext method.

Overall, the hardware information extraction process and the saveResultOpenNext function are crucial components of the hardware documentation process.

### 3.2.5 Saving to Excel and Cleanup

The PDF to Excel conversion process is a crucial component of the hardware documentation process. Once each image has been reviewed successfully, the label above the page list discloses this, and the "Export PDF to Excel" button can be pushed successfully. However, there are certain criteria set in place to prevent a premature handling of incomplete data from the called function. These criteria must fulfill the conditions, including but not limited to having an active Result Window and not having any missing loop file.

```

1     def convertLoopExcel(self):
2         if self.curWorkingResultFileName is None:
3             self.buttonWindow.info_label.setText("Please have an active
    Result Window")
4             return
5         if self.curResultWindow.mainWindow.count() == 0:
6             self.buttonWindow.info_label.setText("Please have an active
    Result Window")
7             return
8         if not os.path.exists(os.path.join(BASEDIR, FINISHED)):
9             os.makedirs(os.path.join(BASEDIR, FINISHED))
10        try:
11            files_processed = [i for i in os.listdir(os.path.join(self.
    directoryPath, self.curWorkingFileNamePage)) if i.endswith(".txt")]
12        except FileNotFoundError:
13            self.buttonWindow.info_label.setText("No page have been
    processed")
14
15        return
16        pages_names = [i.split(".")[0] for i in os.listdir(os.path.join
    (self.directoryPath, self.curWorkingFileNamePage, PDF_PAGES)) if i.
    endswith(".png")]
17        if len(files_processed) != len(pages_names):

```



```

18         self.buttonWindow.info_label.setText("Not all pages have
        been processed")
19         return
20         wb = Workbook()
21         ws = wb.active
22         ws.append(self.curResultWindow.get_header_loops())
23         for i in files_processed:
24             with open(os.path.join(self.directoryPath, self.
        curWorkingFileNamePage, i), "r") as f:
25                 reader = csv.reader(f, delimiter=";")
26                 for row in reader:
27                     ws.append(row)
28         wb.save(os.path.join(BASEDIR, FINISHED, f"{self.
        curWorkingFileNamePage}.xlsx"))
29         self.buttonWindow.info_label.setText("Exported loops to Excel")
30         self.directoryWindow.pages_label.setText("Pages")
31         self.directoryWindow.directoryList.clear()
32         pdf_to_delete = self.directoryWindow.pdfList.item(self.
        directoryWindow.pdfList.currentRow())
33         shutil.rmtree(os.path.join(pdf_to_delete.data(3), pdf_to_delete.
        data(4)))
34         _ = self.directoryWindow.pdfList.takeItem(self.directoryWindow.
        pdfList.currentRow())

```

Listing 3.9: Code snippet of the convertLoopExcel method.

The `convertLoopExcel` function is responsible for converting the loops to an Excel file. The function verifies an active Result Window and no missing loop file. It loops through the folder structure, finding all text files containing the loops for all the pages. Then an Excel file is created, and all the loops are appended. Each loop takes one row in Excel. The Excel file is named by the PDF's original name.

After the whole process of importing the PDF, converting them to images, extracting text using OCR, searching for the correct word, having the user review the images, and exporting to Excel, the cleanup can begin. The user gets a job well done from the status label while the PDF from the list, along with all the images in the images list, disappear, and the whole underlying folder structure for that PDF is deleted. The process starts anew with the next PDF.

Overall, the PDF to Excel conversion process is a crucial component of the hardware documentation process.

### 3.3 Performance Experiment

	Portion of 50	Portion of 25	Portion of 10
Converter Worker	2.8	2.78	2.92
Image Processor Worker	31.82	32.33	34.42
Both Workers	34.62	34.12	37.33

Table 3.1: Processing times in minutes for the Converter Worker, Image Processor Worker, and both workers combined for portions of 50, 25, and 10 drawings.

An experiment was conducted to determine the processing time of the Conversion Worker and Image Processor Worker while handling a PDF document comprising of 50 drawings. In view of the time-intensive nature of this task, the experiment aimed to examine the impact of splitting a PDF file into smaller portions on the processing time. The performance experiment involved three different portions, including 50 pages at once, 25 pages at a time, and 10 pages at a time. The runtime was measured using the Python function `time.time`, and the results are presented in Table 3.1 and graphically illustrated in Figures 3.8, 3.9, and 3.10.

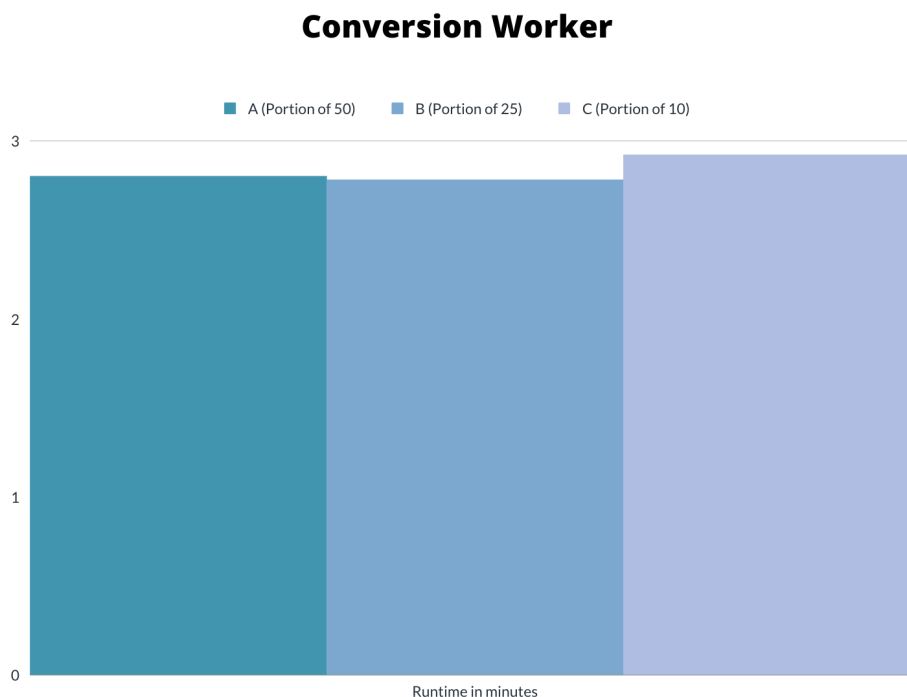


Figure 3.8: Runtime during execution of Conversion Worker.

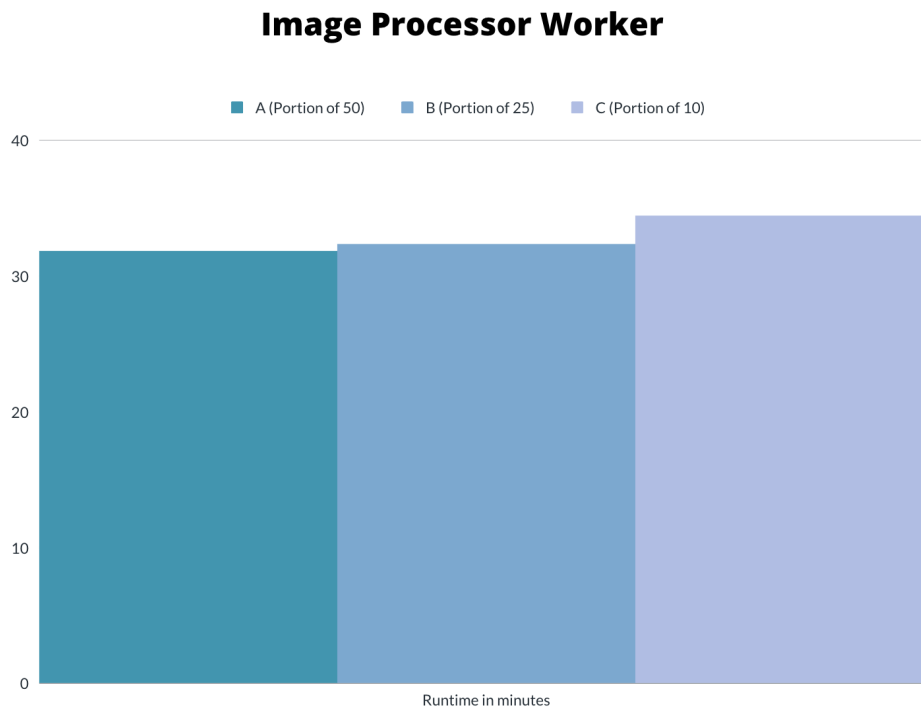


Figure 3.9: Runtime during execution of Image Processor Worker

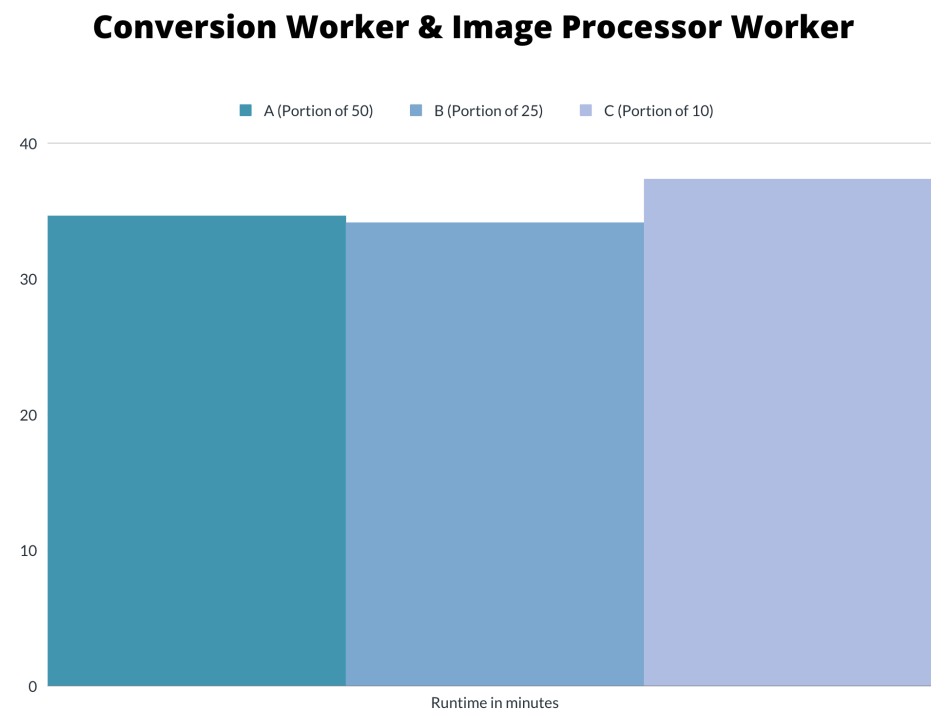


Figure 3.10: Runtime during execution of both Conversion Worker and Image Processor Worker

# Chapter 4

## Discussion and Conclusion

### 4.1 Discussion

Significant strides have been made in improving the user experience by simplifying the workflow through the implementation of various interface changes. The decision to reduce the number of buttons and consolidate their functions' redundant working habits, enables the user to achieve more with fewer clicks. These implementations have resulted in a user-friendly and streamlined application, which enhances productivity and promotes efficiency.

In the developed GUI, the different functionalities are divided into "lines of buttons", each corresponding to a common "line of work". To ensure ease and intuitiveness, each "line of buttons" is color-coded to represent its respective "line of work". Furthermore, to accommodate for the display of large images, the image viewer takes up the majority of the main window. This ensures that the user has a greater overview of the loops. Both successfully and unsuccessfully recognized text is displayed in a separate second window. Additionally, each loop of identified text traverses the width of the second window. This design choice was made to make it easier for the user to compare the identified text across different loops and to ensure that the results are easily interpretable.

The final product consists of several functions, each with a specific task. The "Import PDF" button calls the `importPdfDlg` function, allowing the user to import one or more PDF files. While the next in line, "Process Next PDF" button calls the `getNextPDF` function, providing the user with status updates while processing the PDF. The "Open Result Viewer Window" button calls the `openResultWindow` function, which extracts and presents the result in a second window. Additionally, the "Checked Result, Open Next Page" button calls the `saveResultOpenNext` function, which saves the result and opens a new result in the result window. Two functions are being used in the result window, `textchanged` and `'listelement_update_word'`, that can be called on by writing in the input

field or clicking on an element in the list, updating the correct word with the user's input value. Adding a loop, should the amount be insufficient, the placeholder text will be displayed in red highlighting it is a manually-filled added loop. The `convertLoopExcel` function is called by the "Export PDF to Excel" button, saving all loops to an Excel file and displaying that the file saved successfully.

In regards to the performance experiment, dividing the PDF file into smaller portions did not necessarily result in improved processing time, which is beneficial as it saves users from having to perform extra preparations while using the developed product. However, when both workers were employed, the processing time was slightly reduced when the PDF was divided into two portions, compared to not dividing it at all. This could be due to the fact that there are more pages per PDF that need to be converted into images, as indicated in the performance difference by running only the `Conversion Worker`. Nevertheless, the overall performance difference by dividing the PDF into smaller portions is not significant enough to warrant it as a necessary preparation.

To summarize, the overall design features and underlying functions present a reliable and well designed interface for the project's objective.

## 4.2 Conclusion

In conclusion, this thesis has provided a thorough and effective solution for managing PDF files and delivering the identified information to the user through a well-designed interface. The final product consists of a sequence of steps that include converting the uploaded PDF pages to images, followed by morphological operations for preprocessing. Subsequently, OCR is utilized to extract text-based information from each image in a sequential manner, accompanied by a validation and correction process to enhance the accuracy of the extracted information. Overall, this thesis contributes to the development of more efficient and reliable methods for handling PDF documents.

# Bibliography

- [1] *FactPages: Field Information - Oseberg Sør*, Accessed: 23-March-2023. [Online]. Available: <https://factpages.npd.no/nb-no/field/pageview/all/43506>.
- [2] T. Hansmann, "Harness volatility: Technology transformation in oil and gas," 2022, Accessed: 23-March-2023. [Online]. Available: <https://www.mckinsey.com/capabilities/operations/our-insights/harnessing-volatility-technology-transformation-in-oil-and-gas>.
- [3] *User interface*, Accessed 05-May-2023. [Online]. Available: [Merriam-Webster.com%20Dictionary](https://www.merriam-webster.com/dictionary/user-interface).
- [4] F. Churchville, *User interface (ui)*, Accessed 11-January-2023. [Online]. Available: <https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI>.
- [5] S. Branson, *UX / UI Design: Introduction Guide To Intuitive Design And User-Friendly Experience*. Independently published, 2020, ISBN: 979-8653877315.
- [6] Computer Hope, *Gui*, Accessed 11-January-2023. [Online]. Available: <https://www.computerhope.com/jargon/g/gui.htm>.
- [7] Z. KL, *The command line for first-timers*, Accessed 02-May-2023, 2020. [Online]. Available: [https://dev.to/zoe\\_kl/the-command-line-for-first-timers-2n4d](https://dev.to/zoe_kl/the-command-line-for-first-timers-2n4d).
- [8] *Graphical User Interface (GUI)*, Accessed 11-January-2023. [Online]. Available: <https://www.arimetrics.com/en/digital-glossary/graphical-user-interface-gui>.
- [9] S. B. T. Shu, *GUI Design*. SendPoints, 2015.
- [10] J. Chen, Z. Xing, C. Chen, and X. Xu, "Object Detection for Graphical User Interface: Old Fashioned or Deep Learning or a Combination?," 2020. DOI: [10.1145/3368089.3409691](https://doi.org/10.1145/3368089.3409691).
- [11] *Digital image processing*, Accessed 30-March-2023. [Online]. Available: <https://www.educba.com/digital-image-processing/>.

- [12] University of Tartu. “Digital image processing.” Accessed 30-March-2023. (2004), [Online]. Available: <https://sisu.ut.ee/imageprocessing/book/1>.
- [13] N. Kumar. “Digital image processing basics.” Accessed 30-March-2023. (2023), [Online]. Available: <https://www.geeksforgeeks.org/digital-image-processing-basics/>.
- [14] National University of Singapore. “Image processing tutorial.” Accessed 30-March-2023, Department of Electrical and Computer Engineering. (2003), [Online]. Available: <https://crisp.nus.edu.sg/~research/tutorial/tmp/image.htm>.
- [15] *Digital images — Encyclopedia.com*, Accessed 02-April-2023. [Online]. Available: <https://www.encyclopedia.com/computing/news-wires-white-papers-and-books/digital-images>.
- [16] Digamation, *Understanding bit depth*, Accessed 30-March-2023, 2008. [Online]. Available: <https://digamation.wordpress.com/2008/07/18/understanding-bit-depth/>.
- [17] OpenCV, *About opencv*, Accessed 02-May-2023. [Online]. Available: <https://opencv.org/about/>.
- [18] Python Geeks, *What is OpenCV?* Accessed 02-May-2023. [Online]. Available: <https://pythongeeks.org/what-is-opencv/>.
- [19] P. Soille, *Morphological Image Analysis, Principles and Applications*. Springer, 1999. DOI: <http://dx.doi.org/10.1007/978-3-662-03939-7>.
- [20] MathWorks. “Types of morphological operations.” Accessed 10-May-2023, MathWorks. (2023), [Online]. Available: <https://www.mathworks.com/help/images/morphological-dilation-and-erosion.html>.
- [21] P. Chhikara. “Understanding Morphological Image Processing and Its Operations.” Accessed 09-May-2023. (2022), [Online]. Available: <https://towardsdatascience.com/understanding-morphological-image-processing-and-its-operations-7bcf1ed11756>.
- [22] A. Munshi. “Morphological image processing operations: Dilation, erosion, opening, and closing with and without inbuilt cv2 functions.” Accessed 09-May-2023, Medium. (2020), [Online]. Available: <https://medium.com/@ami25480/morphological-image-processing-operations-dilation-erosion-opening-and-closing-with-and-without-c95475468fca>.
- [23] V. L. Fox and M. Milanova, “Natural image segmentation using morphological matematics and fuzz logic,” 2013. DOI: [10.1109/IRI.2013.6642542](https://doi.org/10.1109/IRI.2013.6642542).
- [24] S. Kulkarni, *Yoda – your only design assistant*, 2019.

- [25] NECC Assistive Technology Center. “What is ocr?” Northern Essex Community College. (Jan. 2018).
- [26] L. Pinto and P. Brito, “A ssd - ocr approach for real-time active car tracking on quadrotors,” 2019. DOI: [10.1007/978-3-030-14070-0\\_65](https://doi.org/10.1007/978-3-030-14070-0_65).
- [27] IGI Global. “Image acquisition.” Accessed: 22-March-2023. (n.d.), [Online]. Available: <https://www.igi-global.com/dictionary/image-acquisition/79991>.
- [28] Q. Technologies, *What is image acquisition in machine vision? - trigger mechanism*, 2021. DOI: <https://qualitastech.com/image-acquisition/image-acquisition-in-machine-vision-trigger-mechanism/>.
- [29] S. Chopra, “Optical character recognition,” *International Journal of Advanced Research in Computer and Communication Engineering*, 2012. DOI: [https://ijarcce.com/wp-content/uploads/2012/03/IJARCCE2G\\_a\\_shalin\\_chopra\\_Optical.pdf](https://ijarcce.com/wp-content/uploads/2012/03/IJARCCE2G_a_shalin_chopra_Optical.pdf).
- [30] S. Reddy. “Pre-processing in ocr!!!” (2019).
- [31] Data Carpentry, *Thresholding*, Accessed: 25-March-2023, n.d. [Online]. Available: <https://datacarpentry.org/image-processing/07-thresholding/>.
- [32] H. Singh, *How images are stored in the computer?* 2021. DOI: [\url{https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/}](https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/).
- [33] R. B. Digvijay Singh and K. Bahuguna, “Matrix data optimization technique applied to bi-faceted binary images for the purpose of modernization,” *Mathematical Statistician and Engineering Applications*, vol. 71, no. 4, p. 15, 2022. DOI: <https://www.philstat.org/index.php/MSEA/article/view/948>.
- [34] O. Onal, G. Ozden, and B. Felekoglu, “A methodology for spatial distribution of grain and voids in self compacting concrete using digital image processing methods,” *Computers and Concrete*, vol. 5, Feb. 2008. DOI: [10.12989/cac.2008.5.1.061](https://doi.org/10.12989/cac.2008.5.1.061).
- [35] P. K. Charles, “A review on the various techniques used for optical character recognition,” *International Journal of Engineering Research and Applications*, 2012. DOI: [https://www.ijera.com/papers/Vol2\\_issue1/DB21659662.pdf](https://www.ijera.com/papers/Vol2_issue1/DB21659662.pdf).
- [36] A. Rosebrock. “Improving ocr results with basic image processing.” (2021).
- [37] N. Alkhalidi. “How optical character recognition algorithms redefine business processes?” Itrex Group. (2022).
- [38] I. Global. “Segmentation.” Accessed: 30-March-2023, IGI Global. (n.d.), [Online]. Available: <https://www.igi-global.com/dictionary/segmentation/26141>.
- [39] A. Alaei, U. Pal, and P. Nagabhushan, “A New Scheme for Unconstrained Handwritten Text-line Segmentation,” 2011. DOI: [10.1016/j.patcog.2010.10.014](https://doi.org/10.1016/j.patcog.2010.10.014).



- [40] M. Javed, P. Nagabhushan, and B. Chaudhuri, "Extraction of Projection Profile, Run-Histogram and Entropy Features Straight from Run-Length Compressed Text-Documents," 2014. DOI: [10.1109/acpr.2013.147](https://doi.org/10.1109/acpr.2013.147).
- [41] S. Reddy. "Segmentation in ocr!!!" (2019), [Online]. Available: <https://towardsdatascience.com/segmentation-in-ocr-10de176cf373>.
- [42] I. Global. "What is word segmentation." Accessed: 30-March-2023. (), [Online]. Available: <https://www.igi-global.com/dictionary/word-segmentation-indo-china-languages/32717>.
- [43] V. Dongre and V. Mankar, "Devnagari document segmentation using histogram approach," 2011. DOI: [10.5121/ijcseit.2011.1305](https://doi.org/10.5121/ijcseit.2011.1305).
- [44] R. S. Kunte, "A simple and efficient optical character recognition system for basic symbols in printed kannada text," 2008. DOI: [10.1007/s12046-007-0039-1](https://doi.org/10.1007/s12046-007-0039-1).
- [45] A. K. Kushwaha, "Analysis of segmentation methods for brahmi scripts," 2019. DOI: [10.14429/djlit.39.2.13615](https://doi.org/10.14429/djlit.39.2.13615).
- [46] J. M. White and G. D. Rohrer, "Image thresholding for optical character recognition and other applications requiring character image extraction," *IBM Journal of Research and Development*, 1983. DOI: [10.1147/rd.274.0400](https://doi.org/10.1147/rd.274.0400).
- [47] C. D. Stefano, A. Marcelli, and A. Iuliano, "A shape-based algorithm for detecting ligatures in on-line handwriting," 2001. DOI: [10.1080/10798587.2000.10642816](https://doi.org/10.1080/10798587.2000.10642816).
- [48] PrintWiki, *Open/closed*, Accessed: 11,-April-2023. [Online]. Available: <http://printwiki.org/Closed/Open>.
- [49] A. Choudhary, R. Rishi, and S. Ashlawat, "A new character segmentation approach for off-line cursive handwritten words," 2013. DOI: [10.1016/j.procs.2013.05.013](https://doi.org/10.1016/j.procs.2013.05.013).
- [50] The AI Learner, *Optical character recognition pipeline – text recognition*, Accessed on: 02-April-2023, 2019. [Online]. Available: <https://theailearner.com/2019/05/29/optical-character-recognition-pipeline-text-recognition/>.
- [51] S. Suganya, "Analysis of feature extraction of optical character detection in image processing systems," *International journal of engineering research and technology*, vol. 3, 2018.
- [52] K. A. Hamad and M. Kaya, "A detailed analysis of optical character recognition technology," *International Journal of Applied Mathematics Electronics and Computers*, 2016. DOI: [10.18100/ijamec.270374](https://doi.org/10.18100/ijamec.270374).
- [53] S. Liao, *An introduction to long short-term memory networks (lstm)*, Accessed on: 02-April-2023, 2022. [Online]. Available: <https://towardsdatascience.com/an-introduction-to-long-short-term-memory-networks-lstm-27af36dde85d>.

- [54] Techopedia, *Long short-term memory (lstm)*, Accessed 02-April-2023, n.d. [Online]. Available: <https://www.techopedia.com/definition/33215/long-short-term-memory-lstm>.
- [55] Klippa. "What is ocr? the ultimate guide to ocr 2023." Accessed 02-April-2023. (2022), [Online]. Available: <https://www.klippa.com/en/blog/information/what-is-ocr/>.
- [56] B. Jang, "Bi-lstm model to increase accuracy in text classification: Combining word2vec cnn and attention mechanism," 2022. DOI: <https://doi.org/10.3390/app10175841>.
- [57] N. Islam, Z. Islam, and N. Noor, "A Survey on Optical Character Recognition System," *TB Journal of Information and Communication Technology*, 2 2016. [Online]. Available: <http://jms.ilmauniversity.edu.pk/index.php/JICT/article/download/641/362>.
- [58] R. Smith, "An Overview of the Tesseract OCR Engine," 2007. DOI: [10.1109/icdar.2007.4376991](https://doi.org/10.1109/icdar.2007.4376991).