VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Robert Harmaakivi

STREAMLINING THE PDF TRANSLATION PROCESS WITH A PYTHON APPLICATION

Tekniikka
2022

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Robert Harmaakivi
Opinnäytetyön nimi	Streamlining the PDF Translation Process with a Python Application
Vuosi	2022
Kieli	englanti
Sivumäärä	44
Ohjaaja	Anna-Kaisa Saari

Opinnäytetyön tarkoituksena oli luoda sovellus, joka toimii työkaluna käännöstöiden apuna. Sovellukseen voidaan syöttää huonolaatuisia PDF-tiedostoja, ja sovellus käyttää konenäköä apunaan parantaakseen tiedoston laatua. Tämän jälkeen teksti luetaan tekstintunnistuksen avulla sekä käännetään toiselle kielelle.

Sovellus toteutettiin Python-kielellä, käyttäen useita siihen kuuluvia kirjastoja, kuten OpenCV:tä, PyTesseractia sekä Googletransia. OpenCV:llä parannetaan tiedostojen laatua poistamalla niistä artefakteja sekä lisäämällä kontrastia. Tämä on olennainen osa PyTesseract-konenäöllä suoritetun tekstintunnistuksen onnistumista. Käännöksen laatu on myös suoraan verrannollinen tekstintunnistuksen onnistumiseen. Graafinen käyttöliittymä toteutettiin käyttäen PyQt-kirjastoa sekä siihen kuuluvaa QtDesigner-sovellusta.

Työn tuloksena on käännöstyötä avustava sovellus, joka helpottaa kääntäjien suorittamaa työtä. Sovellus suorittaa puhdistuksen sekä käännöksen tehokkaasti isoillekin tiedostoille.

Avainsanat konekäännös, konenäkö, PDF, Python, tekstintunnistus

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Robert Harmaakivi
Title	Streamline the PDF translation process with a Python application
Year	2022
Language	English
Pages	44
Name of Supervisor	Anna-Kaisa Saari

The purpose of the thesis was to create an application that works as an assistive tool for translation work. The user can input poor quality PDF files into the software, and the software uses machine vision to improve the quality of the file. The text is then extracted from the file using Optical Character Recognition and translated into another language.

The application was made with Python, using multiple libraries, such as OpenCV, PyTesseract and Googletrans. OpenCV is used to remove artefacts and increase the contrast of the files. This is an essential part of improving the machine reading quality done with PyTesseract. There is a direct correlation between the quality of the machine reading and the degree to which the translation is being rendered in the target language.

The result is an application that can assist the work of translators. Furthermore, the application effectively conducts the cleaning and translation of large files.

Keywords machine translation, machine vision, OCR, PDF, Python

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

The objective of this thesis was to create assistive software to be used by translators. The main goal was to create an application that could clean and translate PDF files that the user could load into the application. Other software in the market fills some of the requirements, usually just the cleaning or the translation. The purpose was to create an application that would do everything needed by the translator in one simple to use package.

One of the greatest hindrances that translators face in their work is the bad quality of the source material. They often receive files for translation, which have been scanned and copied multiple times, and often have artefacts, such as scanlines, dust speckles, and even stains. These problems cause the files to be hard to read, which costs the translator valuable time, and in the worst-case scenario, might even cause errors in the translation if some parts have become unintelligible.

The other part of the software is the automatic translation of the files. As Google translate has taken strides in the quality of its machine translation, it has become the tool of choice for many translators. It provides a preliminary translation of the file, as it translates the most uncomplicated sentences correctly on the first go. The preliminary translation leaves the human translator more time to focus on the parts of the translation that a machine cannot understand or translate correctly.

The PDF cleaner and translation application was developed to answer these issues. The software was developed using Python and a variety of libraries. The software comes with an installer, and a complete graphical user interface, making the installation and usage of the software as simple as possible.

2 TECHNOLOGIES

The technologies used in this project were chosen on their availability and user-friendliness. Most of the software and libraries used are open-source and have extensive written documentation, making it easier to develop software. Python was chosen as a primary language for the project due to the available libraries, which suited the application well.

2.1 Visual Studio Code

Visual Studio Code is a text editor developed by Microsoft. It was released in 2015 and has since become the most popular tool among programmers, with 71% of programmers reporting using Visual Studio Code[1] (Stackoverflow, 2021). Visual Studio Code is a lightweight program compared to full integrated development environments (IDEs), yet it still retains many features of IDEs, such as automatic code completion with IntelliSense, Git commands and debugging. VS Code is also highly customisable, giving the community full access to create extensions for it.

Visual Studio Code is based on Electron, a framework designed for creating cross-platform applications with native technologies. The program is entirely free and open-source, and it can be run on Windows, Linux, or macOS. Visual Studio Code is a code-centric tool that makes editing code files and folder-based project systems easier. It allows the writing of cross-platform Web and mobile applications using the most popular platforms, such as Node.js and .NET core, with integrated support for many languages and rich editing features, such as IntelliSense finding

[1] The Stack Overflow survey is an annual questionnaire aimed at software developers. The survey has questions ranging from age, location, favourite languages, software and frameworks, all the way to questions on where do you learn, how do you learn, and how many years you have been developing software.

symbol references, quickly reaching a type definition, and much more. (Del Sole, 2019)

Extensions are one of the key features of Visual Studio Code. Tools, languages, code snippets, debuggers, key bindings and themes can be added to the program. Visual Studio Code, in particular, allows the enhancement of the code editor with specialised syntax support, such as code snippets and code refactoring. Visual Studio Code is available to every language and tool on any platform, allowing for a limitless number of development scenarios. (Del Sole, 2019)

2.2 Python

Python is a high-level, object-oriented, open-source programming language designed to optimise development speed. Python is a general-purpose language, but it is often called an object-oriented scripting language because it is commonly used to combine other software components to an application. Python emphasises four key concepts (Lutz, 2001):

1. **Quality.** Python makes it simple to create reusable and maintainable software. It was created to raise development quality requirements in the scripting community. Python's straightforward syntax and well-thought-out architecture almost force programmers to write readable code, essential for software that others may modify. Python is also well-suited to contemporary software reuse methodologies. In reality, developing high-quality Python components that can be used in various situations is nearly effortless.
2. **Productivity.** Python has been designed for speed of development. Python makes it simple to develop applications quickly because the interpreter handles aspects that must be coded directly in lower-level languages. For example, there will be no type definitions, memory management, or build procedures in Python scripts. However, fast initial development is only one

component of productivity. Programmes must develop code that can be executed by a computer and read and maintained by other programmers. Python produces programs that are easy to comprehend long after being developed because its syntax is similar to executable pseudocode[2]. Python also supports advanced paradigms, such as object-oriented programming, which help developers be more productive and reduce development time.

3. **Portability.** Almost any computer system in use today can execute Python programs without modification. Python scripts may now be found on everything from IBM mainframes to Cray supercomputers, as well as notebook PCs and mobile devices. Although some platforms have nonportable extensions, the core Python language and libraries are platform-neutral. For example, most Python programs written on Linux will generally run on Windows right away, and vice versa. Furthermore, a GUI program written with Python's standard Tkinter library will operate natively on Linux, Windows or Macintosh without any source code changes.
4. **Integration.** Python is built to work with a variety of additional tools. As a result, Python programs may easily be mixed with and script other system components. Python scripts, for example, may now use existing C and C++ libraries and communicate with Java classes, among other things. Furthermore, programs written in other languages can just as efficiently run Python scripts by calling C and Java API functions.

Guido van Rossum invented Python in 1990 while working at the Dutch National Research Institute for Mathematics and Computer Science. It was initially designed to be a scripting language for the Amoeba distributed operating system and the ABC language. Python's design proved to be sufficiently broad to cover various fields. Hundreds of thousands of engineers utilise it in increasingly diversified jobs

[2] Pseudocode is a way of representing code, without being written in any programming language

worldwide. Today, Python is used in commercial products to test chips and boards, design GUIs, search the Web, produce videos, script games, provide maps and email over the Internet, and much more. Python's target domains are only restricted by the scope of computers in general because it is an entirely general-purpose language. (Lutz, 2001)

2.3 Pillow

Pillow is a Python library with image processing capabilities. Pillow is a continuation of the original Python Imaging Library (PIL). PIL was developed initially by Fredrik Lundh and was active from 1995 to 2009. After Lundh ceased the development of PIL, Alex Clark took over the project and started developing Pillow from the remainings of PIL and forked it into its own project. Clark has stated that the goal of Pillow is to foster and support the active development of PIL by continuous integration testing, publicised development and regular releases. (Pillow, 2022a)

The Pillow library provides extensive file format support, with over 30 different file formats supported (Pillow, 2022b). An important thing to note is that Pillow uses the metadata of the image to detect the file format instead of using the file extension. Pillow has three main uses:

1. **Image Archives**. Pillow is ideal for batch processing and image archiving. In addition, the library can be used for thumbnail creation, converting between file formats and printing images.

2. **Image Display.** The library comes with functionality from the Tk PhotoImage and BitmapImage interfaces, allowing the display of images directly from the application. There are also functions for using an external display utility.

3. **Image Processing.** The Pillow Library contains image processing functionality, including point operations filtering and colour space conversions. The

library also comes with functions to resize and rotate the images. A histogram method also allows the user to pull statistics out of the image.

All this functionality makes Pillow the preferred technology for the image processing requirements in this project. (Pillow, 2022c)

2.4 Pdf2image

Pdf2image is a Python library that wraps the pdftoppm utility to convert a PDF to a PIL image object (Belval, 2022). Pdftoppm is originally a Linux utility to convert PDF files into Portable Pixmap, Portable Graymap, or Portable Bitmap format (Glyph & Cog LLC, 2011). Pdf2image is a simple way to extract pages from PDF files, to enable their editing in a picture format. After the conversion, the images of pages can be saved separately and then opened and used by other image processing tools, such as Pillow.

2.5 Tesseract and PyTesseract

Tesseract is an open source Optical Character Recognition engine. Optical Character Recognition turns paper documents or text from images into editable and searchable digital documents, such as text files. OCR works by analysing the patterns of light and dark that form the letters and numbers in the image and turning them into text. Early OCR systems worked only with specific fonts explicitly designed to be able to be read by an OCR. Modern OCR software, however, can understand nearly all fonts and, in some cases, even handwriting. (Konica Minolta, 2018)

The Tesseract OCR engine was initially developed by Hewlett-Packard Company, with the project running between 1985 and 1995. In 1995 Tesseract ranked among the top 3 OCRs in accuracy in a test conducted by the University of Nevada in Las Vegas. There was no knowledge outside of HP for the ten years that the project was in development. Tesseract was close to being adopted into the HP scanners to become the key differentiator between HP and their competitors. Tesseract had

a real advantage initially over other OCR engines at the time, as it was more accurate, especially with low-quality images. The disadvantage of Tesseract was its speed and required processing power, meaning it needed to be assisted by a computer, as the scanners of the time did not have enough processing power. All of this, combined with other issues such as localisation, HP decided to discontinue the development of Tesseract. (Smith, 2013)

After 1995, the project laid dormant for ten years, with no development being done on it. In 2004 or 2005, engineers at HP decided to release Tesseract as open-source software with the help of the Information Science Research Institute at the University of Nevada in Las Vegas and Google. In 2006, Tesseract was released as open-source software, being developed further by Google. In the beginning, only the English language was supported, and the software was not as accurate as other commercially available OCRs. Tesseract was still the most accurate open-source OCR available. (Google, 2006)

Five versions have been released for Tesseract throughout the years, each bringing more functions and supported languages. At the moment, Tesseract supports over 100 languages, including ideographic languages, such as Chinese or Japanese, and right to left written languages, such as Hebrew and Arabic. The software also supports multiple output formats, including plain text, hOCR, PDF and TSV. Google stepped down from developing Tesseract in November 2018, and the software is currently completely community-driven. There are also third-party graphical user interfaces and trained language packs developed by the community. (Smith, 2022)

Python-Tesseract, or PyTesseract for short, is a wrapper for Tesseract, making it possible to use Tesseract in Python software. PyTesseract works well with Pillow, as it has full support for the imaging libraries used by Pillow. PyTesseract can also be used as direct scripts, omitting the requirement of saving the result to a file but rather using the result directly in the code in a variable. (Lee, 2022)

2.6 Googletrans

Googletrans is a Python package that allows the usage of the Google Translate API in Python programs for free. It has features such as detecting the language of a text and bulk translations. (Han, 2020) Using this library simplifies the usage of Google Translate API, as it removes the need for a Google Service Account and private keys.

Google Translate is a free machine translation software with over 500 million users worldwide (Turovsky, 2016). Google Translate was initially launched online in 2006, with the possibility to translate texts from English to Arabic and vice-versa. Around the time when Google Translate launched, the go-to method of translation software was to use a rules-based approach, which required much work by linguists to define vocabularies and grammar. Google's approach, however, was to implement a system where they would feed words, phrases, and different kinds of translated texts in both languages. They would then apply statistical learning techniques to build a translation model. (Och, 2006)

In 2016, Google announced they would move Translate from the phrase-based model into a neural network model called Google Neural Machine Translation system. The GNMT brought machine translation closer to human translation (Figure 1). That has become possible primarily due to the advances in machine intelligence: GNMT employs state-of-the-art training techniques to enhance its translation capabilities continuously. The critical difference between a phrase-based model and a neural network-based translation model is how longer texts and phrases are translated. A phrase-based model takes a sentence, breaks it down into words and translates each word individually. However, the neural machine translation considers the entire input sentence as a unit for translation. (Le & Schuster, 2016) The result of this is a more natural-sounding translation, as we humans perceive language in complete sentences rather than individual words.

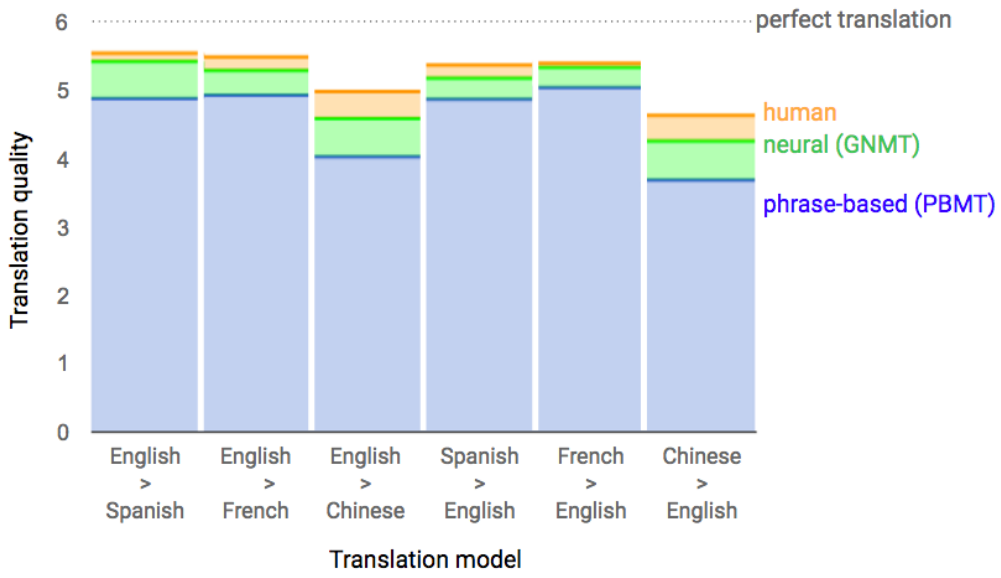

Figure 1. Comparative evaluation of different translation models

In 2019 and 2020, Google improved on the translation models even more with the help of M4 modelling. The M4 model is called a Massively Multilingual Massive Neural Machine Translation. This model uses other languages in the same linguistic family and even from other language families in some cases (Figure 2). That is done to improve the translation quality, especially in low-resource languages, where there is not much data. (Bapna, 2019) With the usage of the M4 model, Google Translate was able to reach an average of +5 BLEU score[3] in all 100+ languages available (Liang Bowen, 2020).

Today, Google Translate is a popular software, with over 500 million users, more than 100 languages, and 100 billion words translated daily. The recent evolution

[3] Bilingual Evaluation Understudy is a popular metric for evaluating machine translations. It is based on a system, in which machine translations are compared to translations of the same texts done by humans.

of Google Translate has also been a mobile application that uses machine vision to directly translate a text when the user points their phone camera at the text. There is also a large community of 3.5 million people who have made contributions to the software, helping Google add new languages and improve the current ones. (Turovsky, 2016)

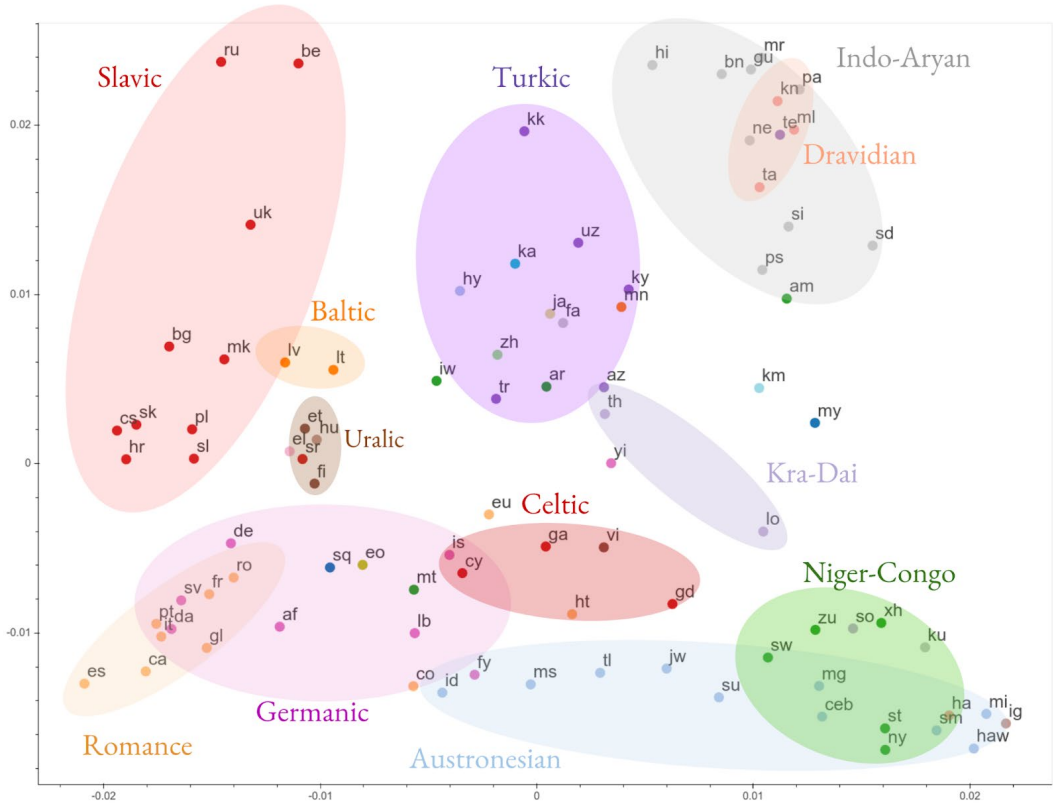

Figure 2. Linguistic families of the available languages in Google Translate

2.7 OpenCV2

OpenCV is a free and open-source computer vision library. The library is created in C and C++ and runs on Linux, Windows, and Mac OS. Interfaces are being actively developed for Python, Ruby, Matlab, and other languages. OpenCV was created with a heavy focus on real-time applications and was designed to be computationally efficient. OpenCV is developed in optimised C and is multicore CPU compatible.

The purpose of OpenCV is to provide a simple-to-use computer vision infrastructure that allows individuals to construct reasonably complex vision applications quickly. Over 500 functions are available in the OpenCV library, covering a wide range of vision topics, including industrial product inspection, robotics, security, user interface, camera calibration, stereo vision, and medical imaging. An entire general-purpose machine learning library is also included with OpenCV. Statistical pattern identification and clustering are the focus of the machine learning library. It is beneficial for the vision problems at the heart of OpenCV, but it is also versatile enough to be applied to any machine learning problem. (Bradski & Kaehler, 2008)

2.8 PyQt

PyQt is a collection of Python bindings for the cross-platform application framework Qt, which combines Qt's and Python's advantages. PyQt allows embedding Qt libraries in the Python code, allowing the creation of graphical user interfaces in Python. In other words, PyQt allows the usage of the Python code to access all of Qt's features. PyQt requires the Qt libraries to function; thus, when installing PyQt, the required version of Qt is also installed on the PC. (Harwani, 2018)

Riverbank Computing develops pyQt. It is more than just a graphical user interface framework. Network sockets, threads, Unicode, regular expressions, SQL databases, SVG, OpenGL, XML, a fully complete web browser, a help system, a multimedia framework, and a comprehensive selection of GUI widgets are all included. Qt classes use a type-safe but loosely connected signal/slot method for communicating between objects, making it simple to construct reusable software components. Qt also comes with Qt Designer, a tool for creating graphical user interfaces. Qt Designer may be used to generate Python code with PyQt. Qt Designer also allows the addition of additional GUI controls written in Python. (Riverbank Computing)

3 DESCRIPTION OF APPLICATION

The purpose of the application is to be used as an assistive tool by translators by cleaning and automatically translating scanned PDF files. The issue is that the files received for translation are frequently in poor condition. The files contain artefacts, such as scan lines and dust speckles, and the files are often skewed. As the files are also scanned, most PDF readers will not allow copying the text from the file. That also causes it to be nearly impossible for online automatic translation software to recognise the text or even try a translation. Many offerings instantly show an error message saying they cannot translate scanned PDFs. There are ways in which one can do all of this software functionality. However, it requires multiple steps through different software. This software aims to combine these solutions into one package that takes a PDF file as an input and provides a translated editable text file as an output.

3.1 Operation

The first step in the application is to open a PDF file. After the PDF is opened, the pdf2image library is used to convert each page of the file into an image. These images are automatically stored in an array by pdf2image, and the images will be saved to separate image files by looping through the array. The purpose of this is to have a reference point to compare the starting point and the cleaned version.

After the PDF file is saved into images, the software opens the first image file, gets the size information of the image, and stores the width and height values into variables. Next, the software goes through each pixel in the picture and gets the RGB value of the pixel. This value is then compared to the preset value or a value set by the user. The pixel colour will be set to black if the value is less than the preset value and white if the value is greater. The former causes the picture to be completely black and white, with no grey values. This binary cleaning is usually sufficient if the files are scanned. If the files were pictures of pages, this method

would not be sufficient, as it might cause some parts of the page to become completely black (Figure 3). A more comprehensive system with proper algorithms can be used in these cases, such as Adaptive Thresholding (Figure 3) or Otsu's Binarization, both provided in the CV2 library. (OpenCV, 2022) After whichever method was chosen and used, the cleaned image is saved.

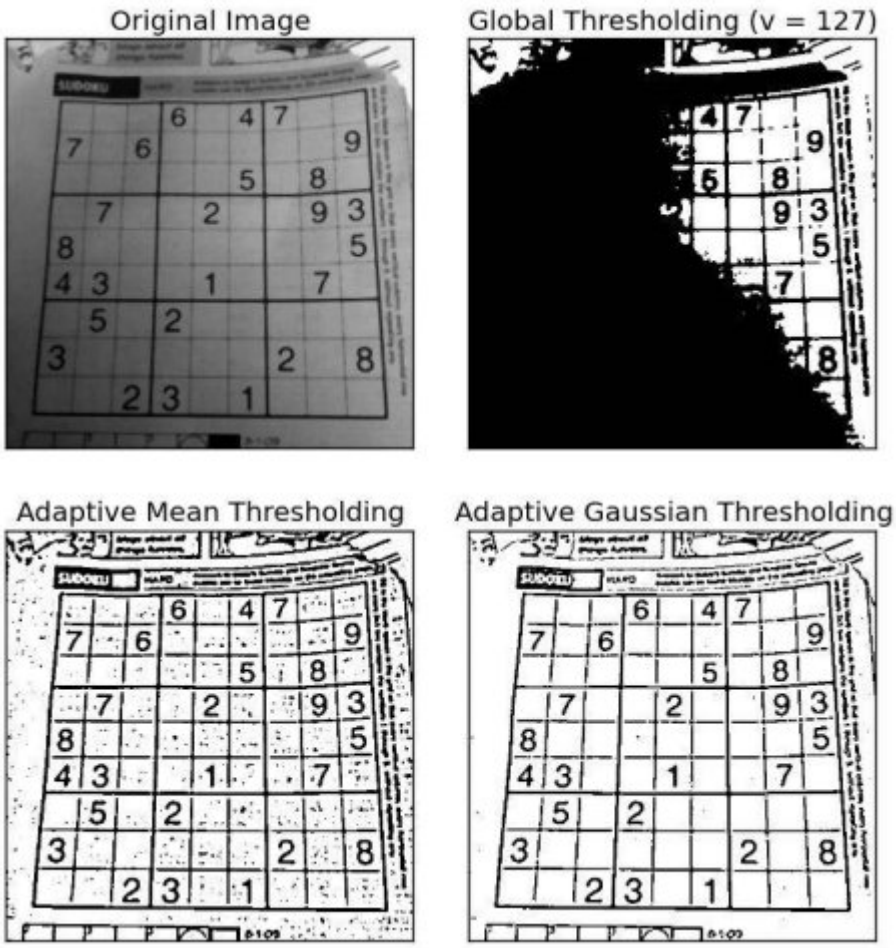

Figure 3. Global thresholding compared to Adaptive Thresholding

After the image is cleaned, it will be opened by CV2 and passed to PyTesseract, to have the text extracted from the images. Tesseract has a better chance of extracting the correct text when the image is first cleaned of all the artefacts. The extracted text will be then saved to a separate text file and appended to a string containing the text from all of the original pages.

After the text has been extracted and saved to a text file, the last thing to do is to pass the text file into Google Translate to be translated. The Googletrans library could be used to detect the input language automatically. However, it was decided to have the input language chosen manually by the user in this case, as it is a safer option: If there are some parts in the text, which is in another language, for example, an abstract in English in the beginning, this might cause issues with the automatic language detection. Therefore, the user also chooses the output language, and Googletrans will use these parameters as a basis for the translation. For translating, Googletrans connects automatically to the Google Cloud Translate API, which implies that the developed software requires an internet connection to work. The translated text is then saved to a text file, and the text is also appended to a string variable, which contains all the translated texts.

After all these steps have been made for a single page, the software will open the following page continuing with the whole process from the beginning. When no more pages are left, the string variables with the texts from all pages will be saved to separate files. The software saves each step into folders for review in the current form because automated procedures, such as machine vision and machine translations, may lead the software to make mistakes. If anything appears not to be correct in the final result, the translator can consult the preceding files to make an informed guess as to what the words are. A complete flowchart of the software is shown in Figure 4

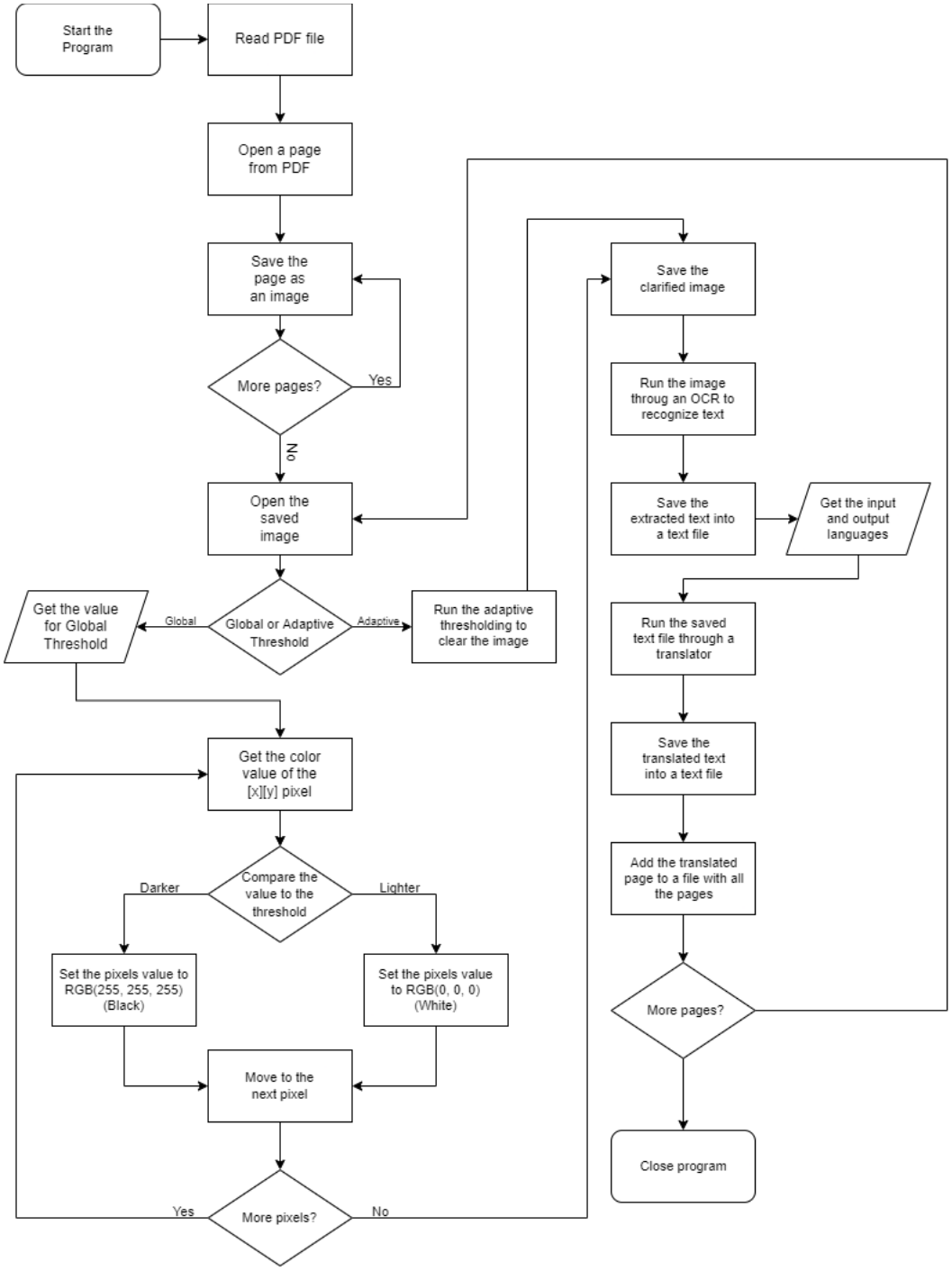

Figure 4. The flowchart of the software

3.2 Usage

Because the output is not ideal, this program should only be used for assistive reasons. The purpose is to get a preliminary translation, speed up, and help the process of translating. Because Google Translate can correctly translate most basic and regularly used sentences, the translator may focus on portions of the text that a machine cannot translate correctly: implied meanings, distinct tones, wordplays or proverbs. The translator's job becomes more manageable with the preliminary translation. With tight deadlines, having the time to focus on the essential aspects is critical.

3.3 Graphical User Interface Design

The GUI was designed to be fairly simple but still offer users enough choices and flexibility to get the best result possible (Figure 5). As the Tesseract optical character recognitions result is directly affected by the quality of the source material, it was essential for the user to have the ability to see the cleaned pages before being passed on to Tesseract. The ability to fine-tune the cleaning process, combined with the view of the original file next to the cleaned one, provides an easy and fast way to get reliable results from the cleaning process.

There is a button in the top left corner of the GUI to open a file. The top left corner is where users look when opening a new application, so it was natural to put the file opening button there. When the button is clicked, a file explorer window will open on the computer, allowing the user to go to the chosen PDF file on their system. (Figure 6).

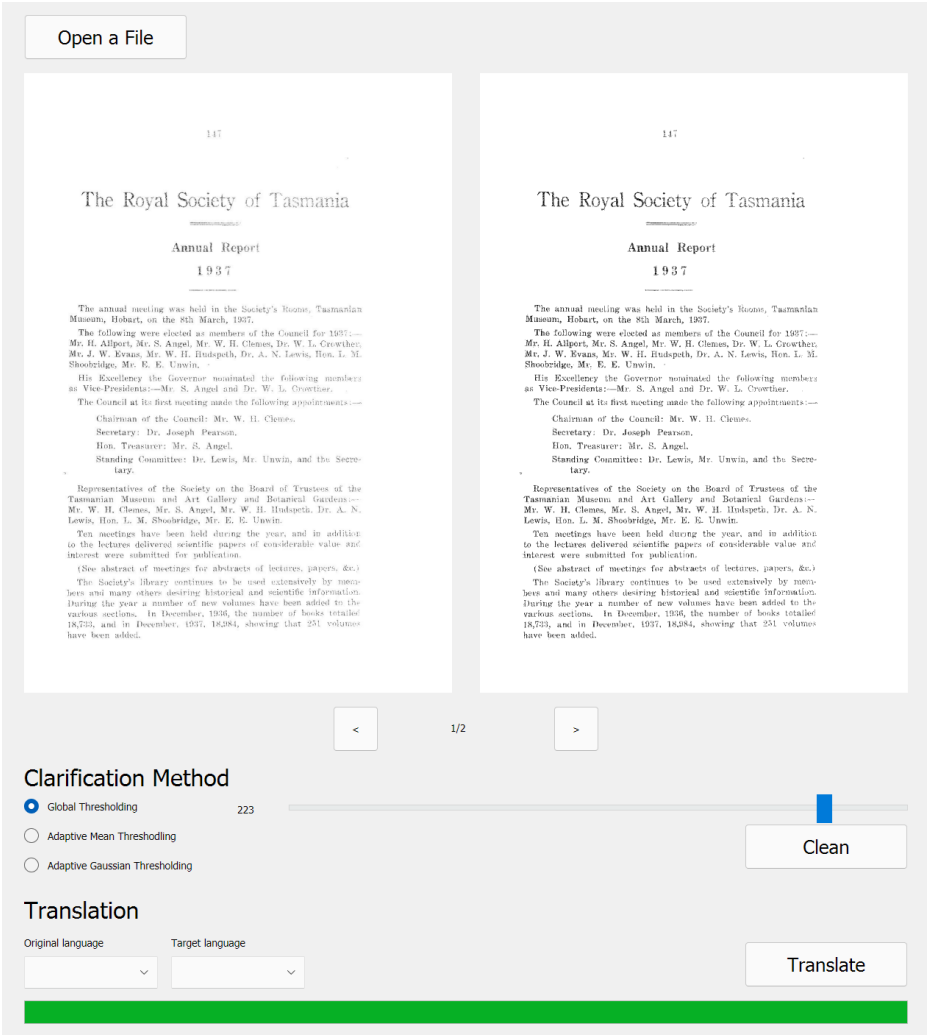

Figure 5. Full GUI

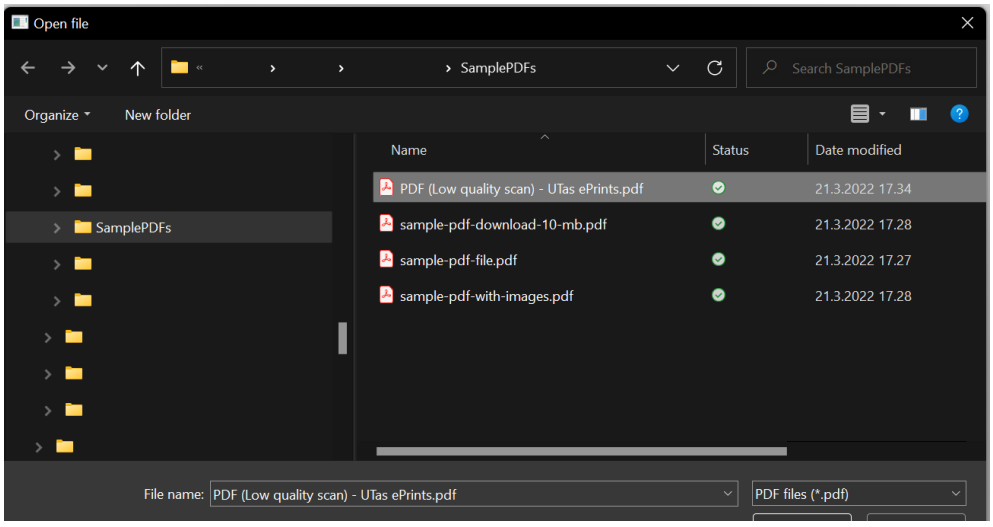

Figure 6. Open file -prompt

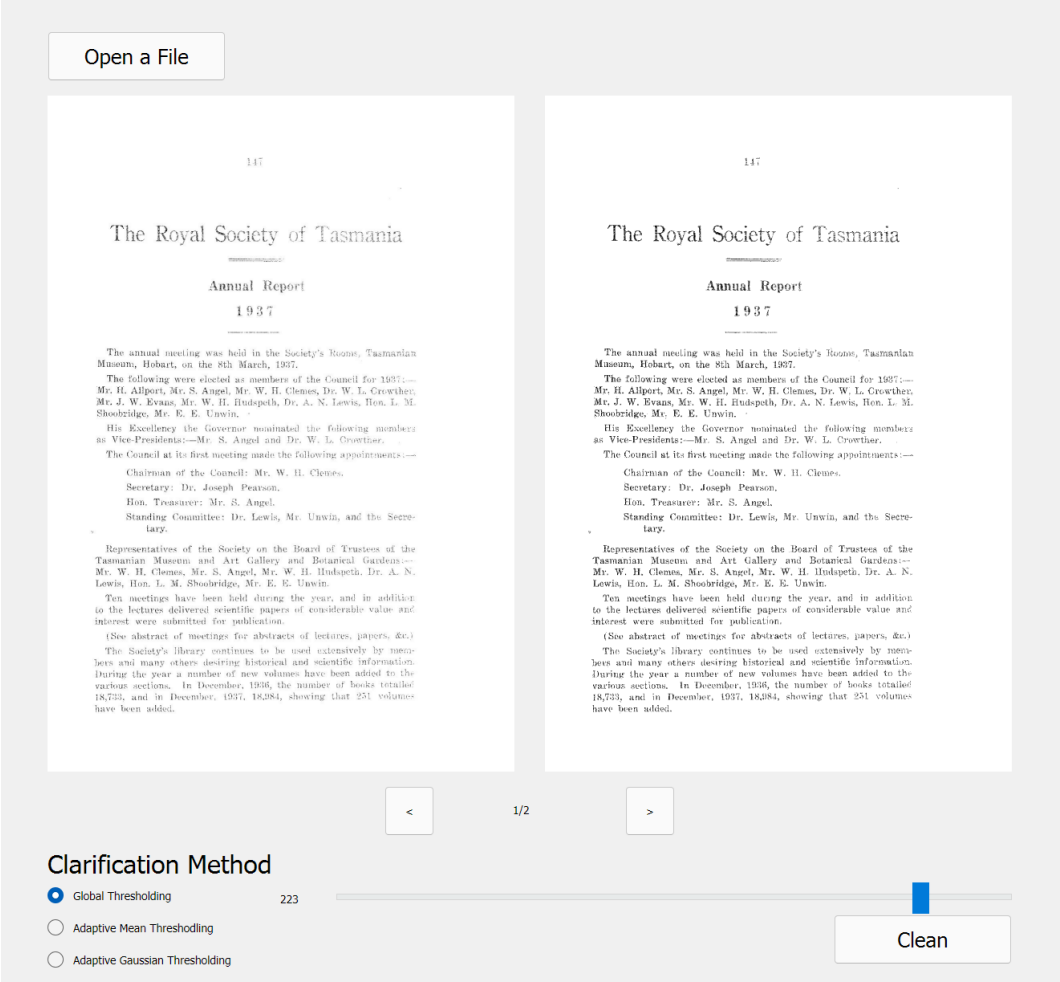

Figure 7. Page comparison and cleaning methods

The last part of the software is the translation options at the bottom. There are two drop-down menus, one for the source language and one for the output language (Figure 8). When the user presses the translate -button, a prompt will open, asking the user where they want the files saved. After this, the actual translation process will begin. Because the process might take even 10 seconds per page, a progress bar was added at the bottom for the user to see the progress of the process (Figure 8).

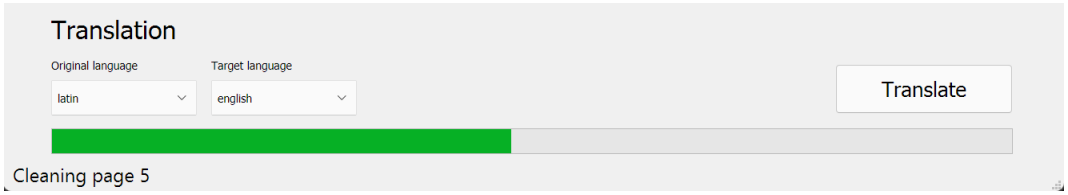

Figure 8. Translation options and progress bar

4 IMPLEMENTATION

The implementation of the software could be divided roughly into three phases. The first phase was the research phase, where the scope of the project and the already available software were studied. In the second phase, a preliminary proof of concept version of the software was developed. The first version was run on the command line and required some technical knowledge and a separate installation of the different software. The final version had nearly the same functionality as the command line version. However, it was made to be far more user friendly with a graphical user interface and an installer to make it as easy as possible for the end-user to start using the software.

4.1 Research

The research phase began with a meeting with the client to hear the desired functions for assistive software for translations. The immediate problem that needed to be solved was the low quality of the scanned PDF files and the inability to copy text from the files. There is already software on the market which makes it possible to turn the text in scanned PDF files into a copyable format. However, most of this software fails to provide satisfactory results because of the low quality of the files. One of the most popular offerings is Adobe Acrobat, which has an automatic enhancement and character recognition function, but it fails to provide accurate results in low-quality files. The result of the enhancement and OCR functions can have trouble distinguishing between scan lines and actual text (Figure 9).

Figure 9. Copied and pasted text from Adobe Acrobat DC Pro

The second feature the client wanted to see in the software was an automatic machine translation to get a preliminary version of the translation. There are,

again, software online, which does this automatically when uploading a PDF file to the software, but they often do not work with scanned PDF files. The result was usually a direct prompt saying the software does not work with scanned PDF files (Figure 10), or the result was utterly unreadable (Figure 11).

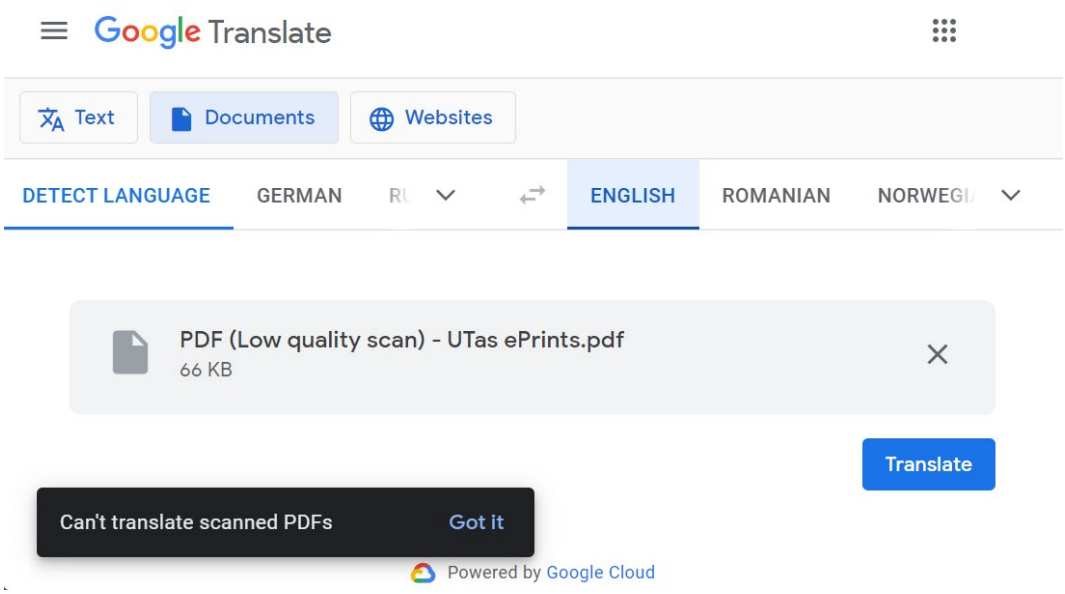

Figure 10. Google document translation

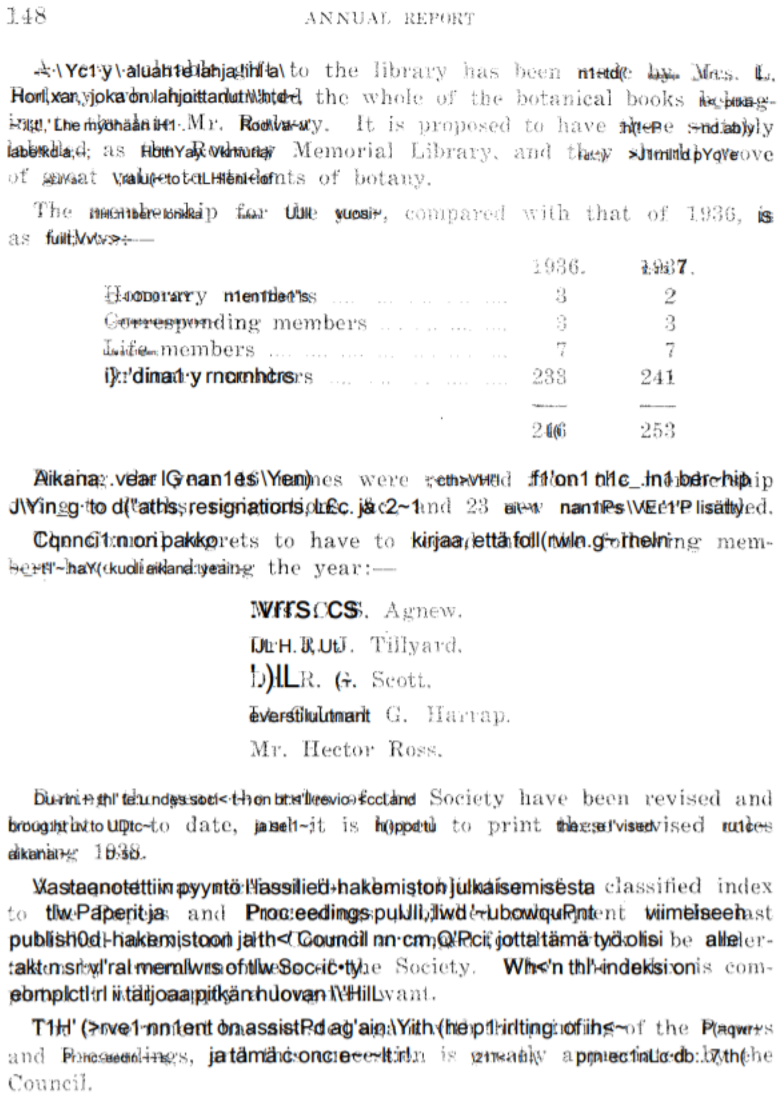

Figure 11. Result of a Multilizer.com automatic translation

4.2 Command-line Version

The proof of concept application development began when enough study was performed and the need for such software was recognised. The first step of developing the software was to find out the best options for a language to write the application. Python was chosen as a language with all the available libraries and ease of use. The whole software in its initial form was written in under 60 lines of code and was fully functional at that point (Figure 12). All the variables were hard-coded, so the filenames needed to be changed in the source code before running the software. The application had only the option of a global thresholding cleaning method, and the value of that was set at 130, as this proved to be the most consistent value with good results. The source and destination languages of the translation also needed to be changed from the source code. They were set at Norwegian to Romanian, as these were the languages of the test files.

The problem with this version was unmistakably its lack of usability. The original thought was to create a more advanced version of the proof of concept, in which the user may provide filenames and languages via the command line. However, because this would have taken too long, the emphasis was on developing a final, graphical version of the software.

```
from PIL import Image
from pdf2image import convert_from_path
from googletrans import Translator
import pytesseract
import cv2

translator = Translator()

im = convert_from_path('full5.pdf')
fulltext = ""
for i in range(len(im)):
    im[i].save('img-original/page' + str(i) + '.jpg', 'JPEG')

for k in range(len(im)):
    image = Image.open('img-original/page' + str(k) + '.jpg')
    pix = image.load()
    width, height = image.size
    i = 0
    j = 0

    while i < height:
        j = 0
        while j < width:
            if(pix[j, i] < (130, 130, 130)):
                pix[j, i] = (0, 0, 0)
            else:
                pix[j, i] = (255, 255, 255)
            j += 1
        i += 1

    image.save('img-clarified/page ' + str(k) + ' clarified.png')

    imageToText = cv2.imread('img-clarified/page ' + str(k) + ' clarified.png')
    text = pytesseract.image_to_string(imageToText)

    text_file = open("txt-original/originaltext" + str(k) + ".txt", "w")
    text_file.write(text)
    text_file.close()

    translated_text = translator.translate(text, src='no', dest='ro').text
    text_file_translated = open(
        "txt-translated/translatedtext" + str(k) + ".txt", "w", encoding='utf-
8') text_file_translated.write(translated_text)
    text_file_translated.close()

    fulltext = fulltext + "\n\nPage " + str(k+1) + "\n\n" + translated_text

text_file_fulltext = open("fulltext.txt", "w", encoding='utf-8')
text_file_fulltext.write(fulltext)
text_file_fulltext.close()

img = Image.open('img-original/page' + str(k+1) + '.jpg')
pix = img.load()
width, height = img.size

image.save('img-clarified/page' + str(k+1) + 'clarified.png')

print("Translation done")
```

Figure 12. Command-Line version of the software

4.3 Graphical Version

After the proof of concept yielded positive results, the effort shifted to creating a graphical user interface for the software and implementing additional, more advanced cleaning methods. The initial step was to set up the cleaning procedures for Adaptive Gaussian Thresholding and Adaptive Mean Thresholding. The CV2 library included a direct method for adaptive thresholds, and both the gaussian and adaptive methods were simply passed as arguments to the method.

```
            # For mean thresholding
            if (self.radioMeanTresh.isChecked()):
                img = cv2.imread('img-original/page' + str(k+1) + '.jpg', 0)
                pix = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
                                            cv2.THRESH_BINARY, 391, 110)

                cv2.imwrite('img-clarified/page' +
                            str(k+1) + 'clarified.png', pix)

            # For gaussian thresholding
            if (self.radioGaussianTresh.isChecked()):
                img = cv2.imread('img-original/page' + str(k+1) + '.jpg', 0)
                pix = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                            cv2.THRESH_BINARY, 391, 110)

                cv2.imwrite('img-clarified/page' +
                            str(k+1) + 'clarified.png', pix)
```

Figure 13. Adaptive thresholding methods

After the adaptive cleaning methods were successfully implemented, the graphical user interface development was started. The choice for the PyQt library was made after thorough research of the available libraries and tools available for Python GUI creation. The decision was made between Tkinter and PyQt, with PyQT providing a more user-friendly approach to designing the user interfaces. The PyQt GUI can be created easily using the Qt Designer software, as the software provides an easy drag-and-drop system for placing the interface elements (Figure 14).

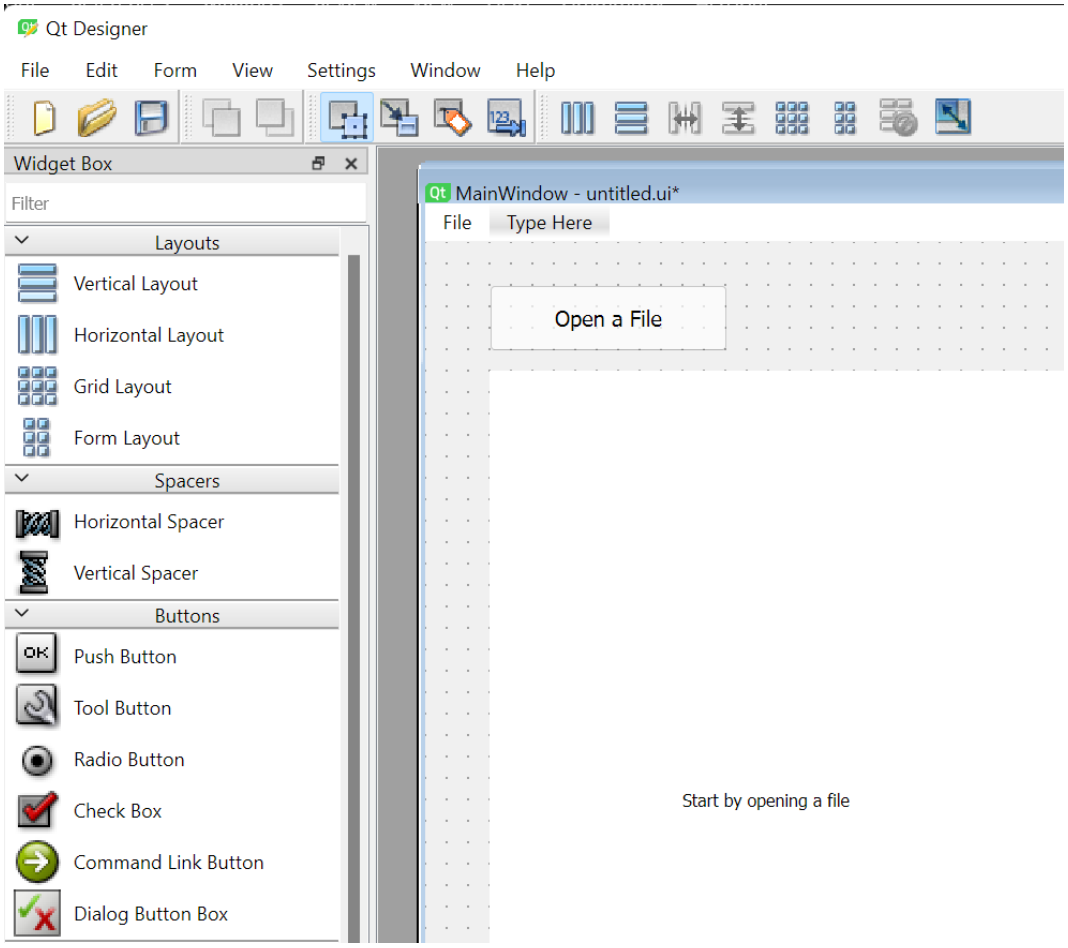

Figure 14. Drag-and-Drop Widget box of the Qt Designer

After an element or widget has been placed, the properties of the element can be adjusted directly in the designer (Figure 15). An essential value in the properties panel is the objectName, as this will be used in the source code when referring to the element, with a findChild method included in the Qt library (Figure 16). The graphical implementation of the GUI was fast and straightforward with the designer tool, and after the designing process was finished, the implementation of the logic was started.

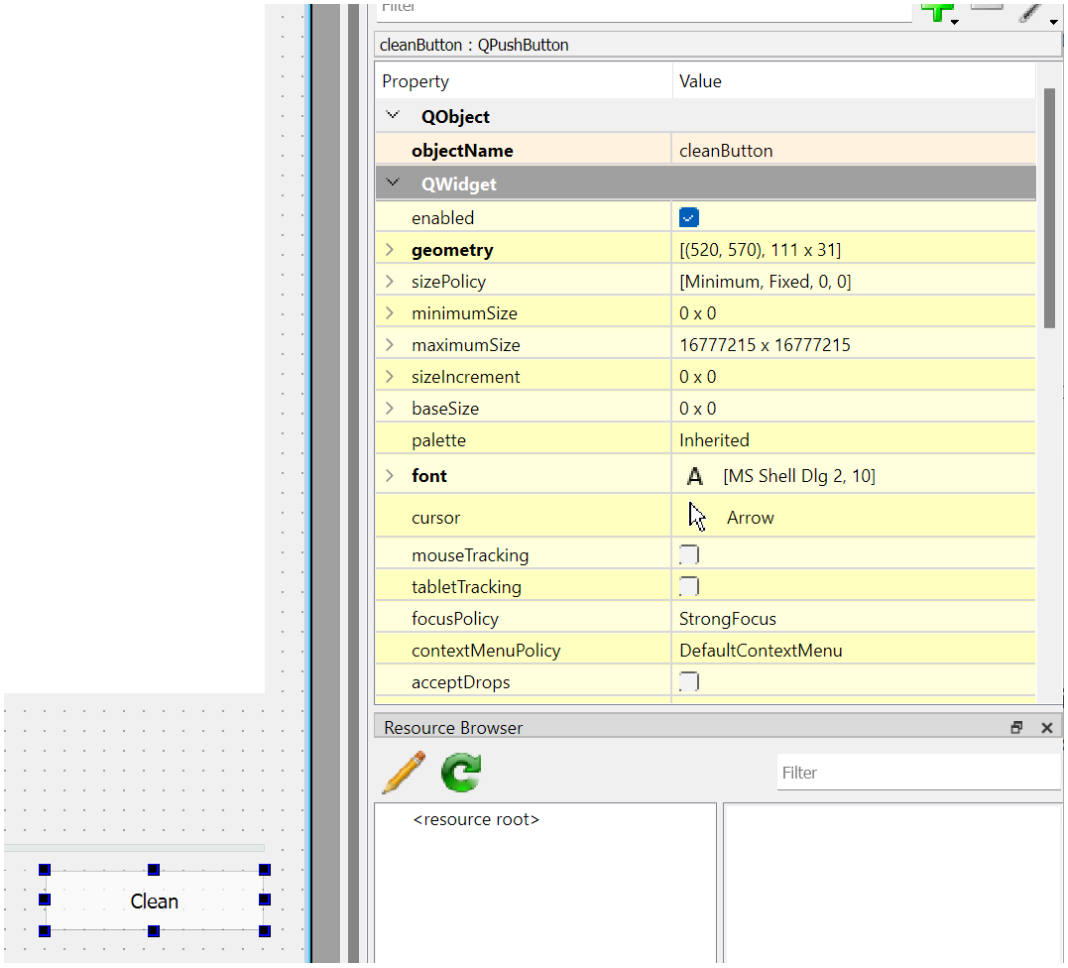

Figure 15. Properties panel of the element

```
self.buttonClean = self.findChild(QPushButton, "cleanButton")
self.buttonLeft = self.findChild(QPushButton, "previousPage")
self.buttonRight = self.findChild(QPushButton, "nextPage")
self.originalFile = self.findChild(QLabel, "originalFile")
self.clarifiedFile = self.findChild(QLabel, "clarifiedFile")
self.labelPageNumber = self.findChild(QLabel, "pageNumber")
self.globalSlider = self.findChild(QSlider, "globalSlider")
self.globalValue = self.findChild(QLabel, "globalValue")
```

Figure 16. findChild method for referencing the GUI elements

The main program consists of two files, the primary Python file and the .ui file, provided by the Qt Designer software. In the source code, the first thing that needs to happen is initialising all the necessary elements of the software (Figure 17.) The whole software is under the UI class, and the QmainWindow, which is the main window of the software, is passed in as an argument. The init method is where the initialisation of the GUI happens. The first thing in the initialisation is loading the .ui file received from the designer. After this, the available languages are loaded from a JSON file into the originalLanguage and targetLanguage combo box lists. The JSON file has key-value pairs of all the available languages in the Google translate API, as the API accepts only language codes, such as 'en' or 'fi', as arguments. The file provides a "conversion" from the language codes to full country names in the drop-down menus for user-friendliness and readability.

```
1  class UI(QMainWindow):
2      def __init__(self):
3          super(UI, self).__init__()
4
5          uic.loadUi("PdfCleaner.ui", self)
6
7          with open("languages.json") as jsonFile:
8              jsonObject = json.load(jsonFile)
9              jsonFile.close()
10
11         for i in googletrans.LANGUAGES:
12             self.originalLanguage.addItem(jsonObject[i], i)
13             self.targetLanguage.addItem(jsonObject[i], i)
```

Figure 17. Initialising the software

After the initialisations, all the elements placed in the designer are then referenced. All the buttons and sliders will also have a connect method to run through them, which will connect the action with a set method.

There are six methods in the software: openFile, leftClicker, rightClicker, sliderChange, cleanClicker and translateClicker. The openFile method allows selecting and opening the desired PDF file from the file explorer. When the open file button is pressed, the file explorer windows opens and shows only PDF files available. After the file is chosen, the PDF file is loaded into the software. The first page is loaded into the space reserved for it. The left and right clicker methods provide a way to navigate the different pages of the file.

The cleanClicker method checks the radio buttons for which of the three cleaning methods is chosen, runs the cleaning for all the pages in the file, and shows the result in the right space. The cleanClicker method also utilises the sliderChange method to get the value for the global thresholding method. The translateClicker method runs the pages first through the Tesseract OCR to read the text in the file. The entire scanned file is first saved as a PDF file, which will have the text in a copyable format. After this, the software takes the input and output languages from the drop-down choices, saves the original text in a separate .txt file and runs through the Google Translate API. Finally, the result of the translation is saved in another .txt file. The user is left with three different files, the PDF file in a copyable format, the original language text file, and the translated text file.

4.4 Installer

The installer was created by first producing a Windows executable file from the Python source code. The file was built using the pyinstaller, which creates a packaged version of Python and all of the required libraries and files to run the software. The pyinstaller generated a file that may be launched without the requirement for the user to install Python. The drawback was the size of the software, which was unreasonably big, almost 100 megabytes. The size was a calculated trade-off for the ease with which the software could be run.

The actual installer was then created by using jrsoftware's Inno Setup. The software comes with an easy to use wizard, which allows the user to choose what files

to include, add licenses and folder structures (Figure 18). The result is a one-click installer, which includes all the necessary files, folders, and an executable in the right place. Custom icons for the software may also be added.

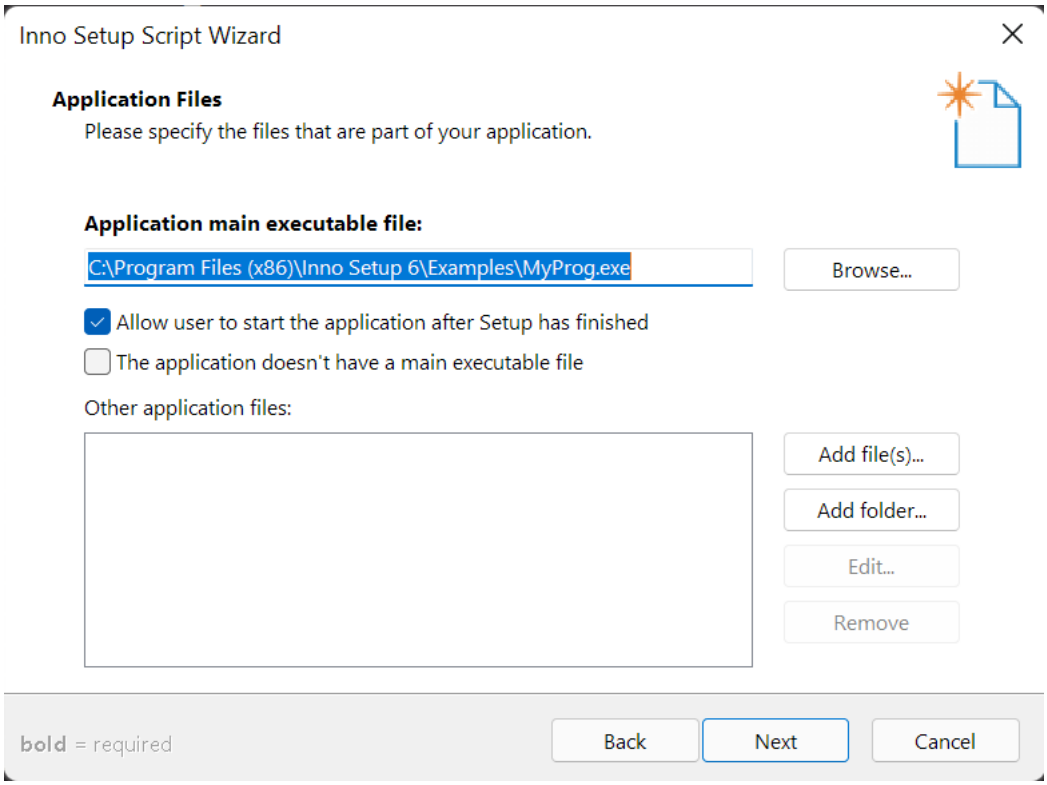

Figure 18. Inno Setup Wizard

5 TESTING

The tests for the software were done to evaluate the functionality of the full software. The tests were run on the installer, the file handling, the actual functions and buttons of the software, the cleaning process, and the translation.

5.1 File handling

After the installation was successful, the file handling and all the functions of the software were tested. The file opening and loading in the software worked as intended, except for a bug that crashes the software if no file is chosen and the file picker window is closed. This could be easily solved by having a conditional for checking if a file was chosen. The progress bar at the bottom of the application worked effectively for smaller files. However, on larger files, the software went into an unresponsive state, the window greyed out, and the progress bar could not show the actual progress but instead jumped to the end when the loading was finished (Figure 21). This issue could have been solved by separating the threads on which the functions and the GUI run, but it would have required a lot more knowledge on how to manage multithreaded applications.

5.2 Cleaning

For the cleaning, a variety of files were tested for functionality. The first file tested was a picture taken from a paper with a shadow placed on purpose to show the issue with the global thresholding method. Next, the same file was cleaned with the adaptive thresholding method, which showed a better, albeit a bit, washed-out result (Figure 22).

The next test was chosen to show the issues with the adaptive thresholding methods. If there was a picture on the same page with text, the adaptive methods struggled to retain the picture intact. However, this was a perfect scenario where the global thresholding method could be used to still retain the picture in the cleaned file (Figure 23).

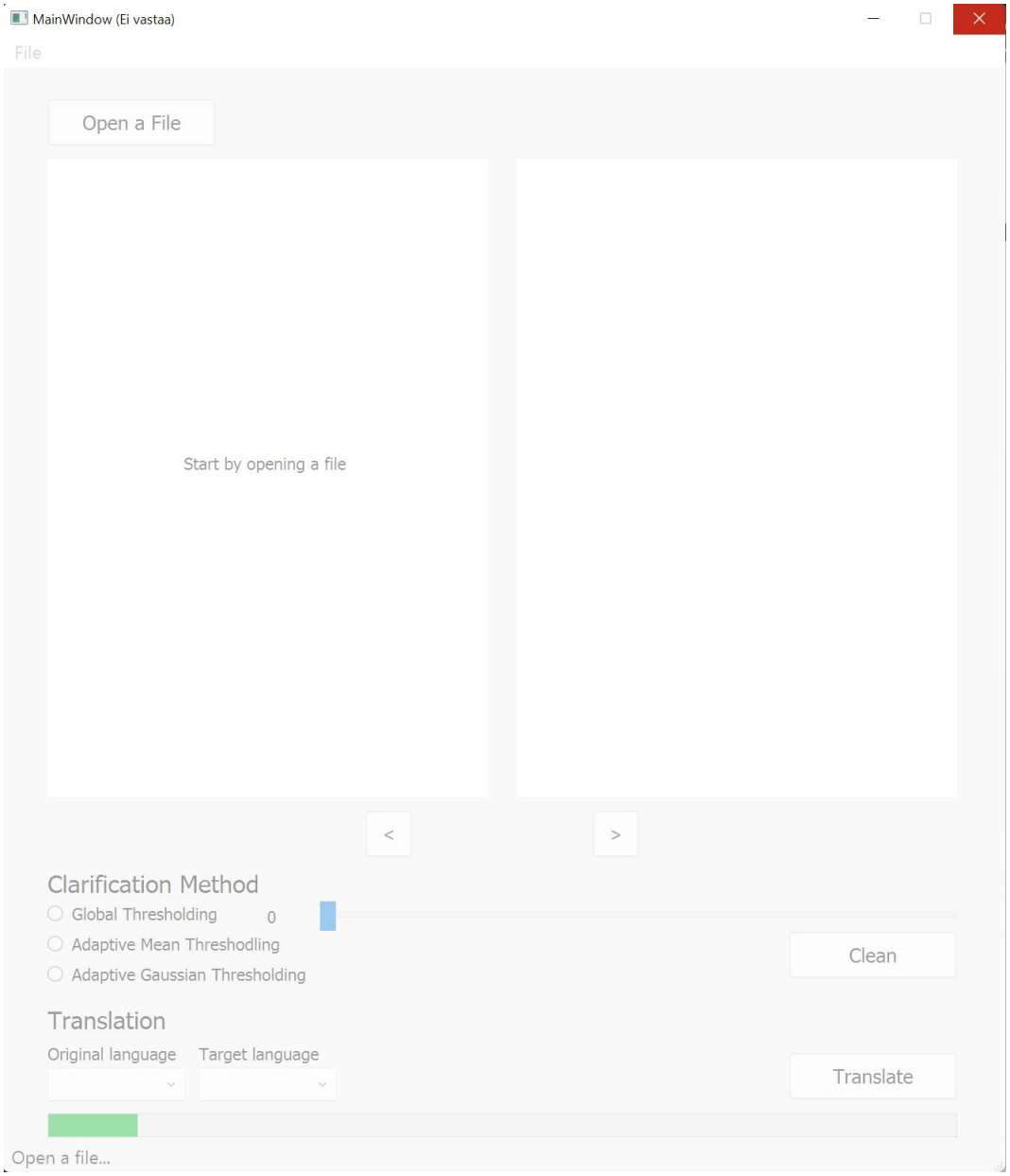

Figure 19. Application in a frozen state

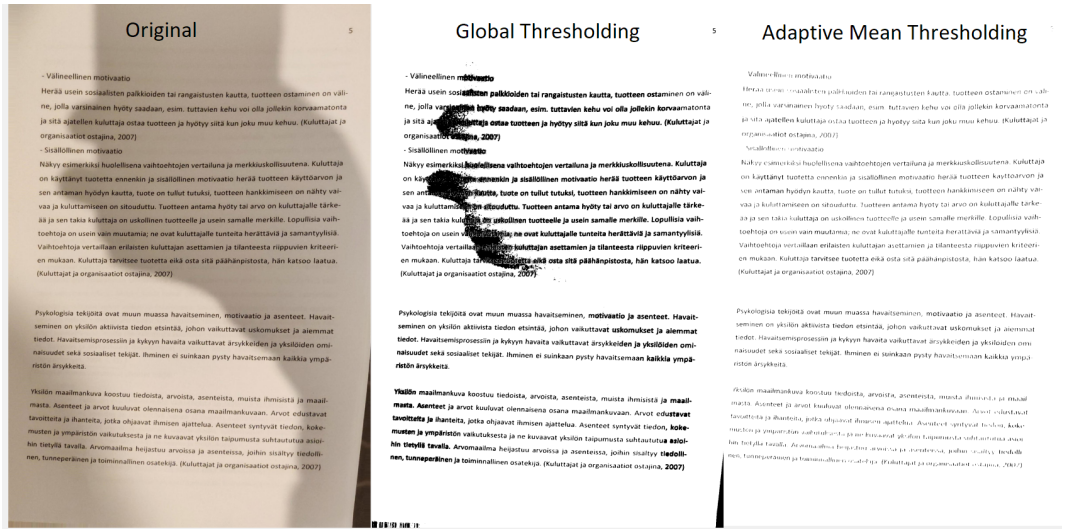

Figure 20. Global thresholding compared against adaptive mean thresholding

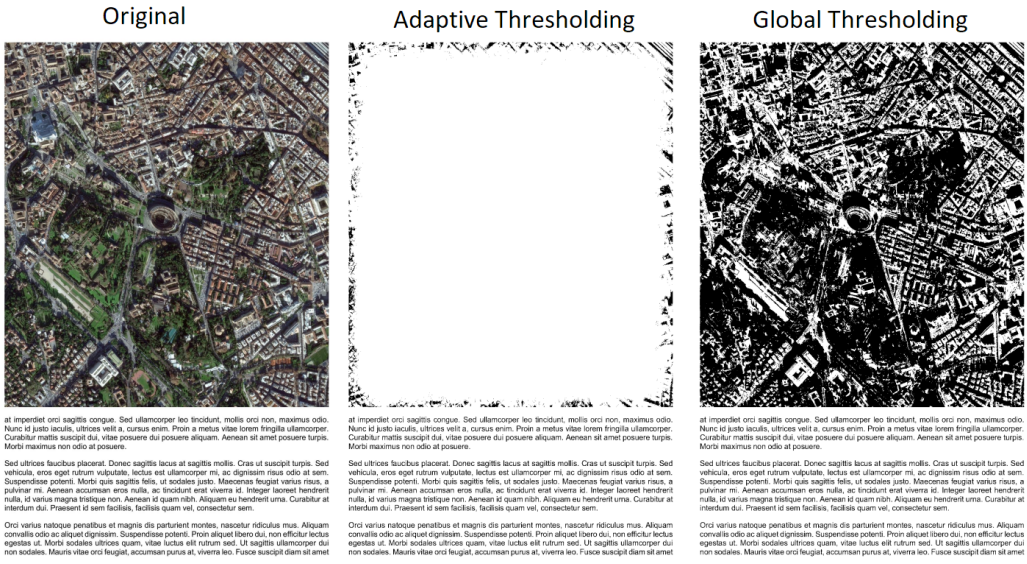

Figure 21. Different cleaning methods with a picture

The final test demonstrated the minor differences between the various cleaning methods in a regular file, with no particular circumstances, but rather a regular scanned text file (Figure 24). The Gaussian cleaning method was usually the best in a standard file like this in terms of the following step in the process, optical

character recognition. Even if the variations appear negligible to the naked eye, there might be a significant difference for the machine reading.

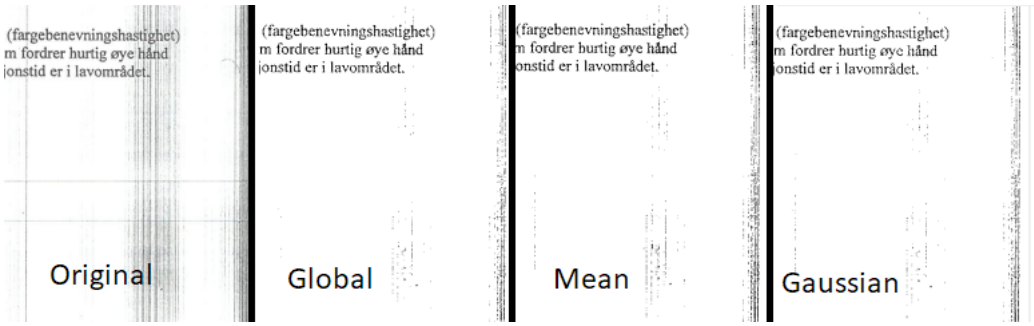

Figure 22. Comparison between all the cleaning methods

5.3 Translation

The translation tests were conducted on files of different quality to assess the complete process of the software. The files chosen were either in Finnish or English, as these are languages that could be analysed appropriately regarding the actual quality of the translation. The first file chosen was the same as shown in Figure 22. The actual text extraction worked even better than expected, with even all the umlauts being retained. The translation in this file worked well on the parts where the sentences were whole. The only issue could be seen in the edges, where the words get split up with the hyphen. In these cases, the software did not realise the words were the same and instead tried to translate both parts of the words separately. This issue could be seen best with the word "Havait-seminen" being translated to "Observe-Simeman" instead of "detection" (Figure 25).

```
Copied from PDF:

Psykologisia tekijöitä ovat muun muassa havaitseminen, motivaatio ja asenteet. Havait-
seminen on yksilön aktiivista tiedon etsintää, johon vaikuttavat uskomukset ja aiemmat
tiedot. Havaitsemisprosessiin ja kykyyn havaita vaikuttavat ärsykkeiden ja yksilöiden omi-
naisuudet sekä sosiaaliset tekijät. Ihminen ei suinkaan pysty havaitsemaan kaikkia ympä-
ristön ärsykkeitä.



The Original text saved in a text file:

Psykologisia tekijöitä ovat muun muassa havaitseminen, motivaatio ja asenteet. Havait-
seminen on yksilön aktiivista tiedon etsintää, johon vaikuttavat uskomukset ja aiemmat
tiedot. Havaitsemisprosessiin ja kykyyn havaita vaikuttavat ärsykkeiden ja yksilöiden omi-
naisuudet sekä sosiaaliset tekijät. Ihminen ei suinkaan pysty havaitsemaan kaikkia ympä-

ristön ärsykkeitä.



The translated text:

Psychological factors include detection, motivation and attitudes. Observe-
Simeman is an individual's active information search that affects beliefs and past
information. The detection process and the ability to detect an impact on stimuli and individuals
signs and social factors. Man is by no means unable to detect all

risen stimuli.
```

Figure 23. The quality of the translation

The second test was run on an English file being translated to Finnish. The chosen file can be seen in Figure 7. The quality of the file was relatively poor because it was severely faded. The quality of the file proved to be difficult for the software, but it could still manage to get a reasonably good result in the end. The only issue was with the actual translation, as Finnish is a complex language to translate. Therefore, the actual quality of the translation is not that great, and the last sentence was not translated at all (Figure 26.)

```
Copied from PDF:

A request was received fou the publication of a classified index
to the Papers and Proceedings published subsequent to the last
published index, and the Cotincil arrangéd for this work to be undertaken
by several members of the Society. When the index is completed
# will supply a long-telt, want.


The original text saved in a text file:

A request was received fou the publication of a classified index
to the Papers and Proceedings published subsequent to the last
published index, and the Cotincil arrangéd for this work to be under-
taken by several members of the Society. When the index is com-
pleted # will supply a long-telt, want.


The translated text:

Saimme pyynnön luokitellun hakemiston julkaisemisesta
lehdet ja menettelyt julkaistiin viimeisen
Julkaistu indeksi ja neuvosto järjesti tämän työn,
useat yhteiskunnan jäsenet. Kun indeksi on
pleted # will supply a long-felt, want.
```

Figure 24. English to Finnish translation

6 SUMMARY

The result of the software ended up being an excellent prospect for future development. The project was hindered by the lack of knowledge of Python and all the relevant technologies but proved to be a great learning experience. With very little prior knowledge of anything connected to this project, there is still a lot to be learned, and the development of the application will continue after the thesis.

While developing the application, many new features were discovered which could be added to the software. The client also requested many new features and improvements after using the final product. Some of the features for further development will be having two different views, an easy-mode view and an advanced view. In easy-mode, the cleaning could be done automatically by a method chosen by the software. With the advanced mode, more options for tweaking the cleaning could be added. Tooltips would also need to be added for all the features. Automatic de-skewing of the files would also be an excellent addition, improving the OCR quality.

Other features for future development will be having different cleaning settings for separate pages, usage of picture file formats, and the possibility to detect the source language automatically. Other more significant features will be having the translation done directly on the PDF to retain all the positioning of the text in the file. Another option would be the automatic creation of a Word file, which would have the text positioning, tables, and pictures in the correct form.

All-in-all, this was a great experience and has taught a lot about the process of creating a complete application for an actual use case. The development will continue onwards, maybe even to a commercial product.

REFERENCES

Bapna, A. 2019. Googleblog. Accessed 06.02.2022 https://ai.googleblog.com/2019/10/exploring-massively-multilingual.html

Belval 2022. Github. Accessed 05.02.2022 https://github.com/Belval/pdf2image

Bradski, G. & Kaehler, A. 2008. Learning OpenCV, Computer Vision with the OpenCV Library. Sebastopol. O'Reilly Media, Inc.

Riverbank Computing. Accessed 27.03.2022 https://riverbankcomputing.com/software/pyqt/intro

Glyph & Cog LLC 2011. Mankier. Accessed 05.02.2022 https://www.mankier.com/1/pdftoppm

Google 2006. Announcing Tesseract OCR. Accessed 05.02.2022 https://web.archive.org/web/20061026075310/http://google-code-updates.blogspot.com/2006/08/announcing-tesseract-ocr.html

Han, S. 2020. Googletrans Documentation. Accessed 06.02.2022 https://py-googletrans.readthedocs.io/en/latest/

Harwani, B. 2018. Qt5 Python GUI Programming Cookbook: Building responsive and powerful cross-platform applications with PyQt. Birmingham, UK: Packt Publishing Ltd.

Konica Minolta 2018. Accessed 05.02.2022 https://www.konicaminolta.com.au/news-insight/blog/how-optical-character-recognition-works

Lee, M. 2022. Pypi. Accessed 05.02.2022 https://pypi.org/project/pytesseract/

Bowen, L. & Caswell, I. 2020. Googleblog. Accessed 06.02.2022 https://ai.googleblog.com/2020/06/recent-advances-in-google-translate.html

Lutz, M. 2001. Programming Python, 2nd edition. Sebastopol. O'Reilly media.

Och, F. 2006. Googleblog. Accessed 05.02.2022 https://ai.googleblog.com/2006/04/statistical-machine-translation-live.html

OpenCV, 2022. Accessed 06.02.2022 https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

Pillow 2022a. About. Accessed 05.02.2022 https://pillow.readthedocs.io/en/stable/about.html

Pillow 2022b. image-file-formats. Accessed 05.02.2022 https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html

Pillow 2022c. Overview. Accessed 05.02.2022 https://pillow.readthedocs.io/en/stable/handbook/overview.html

Python. Accessed 04.02.2022 https://www.python.org/doc/essays/blurb/

Quoc V. & Schuster, M. 2016. Googleblog. Accessed 06.02.2022 https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html

Smith, R. 2013. History of the Tesseract OCR Engine: What Worked and What Didn't. Proceedings of SPIE.

Smith, R. 2022. Github. Accessed 05.02.2022 https://github.com/tesseract-ocr/tesseract

Del Sole, A. 2019. Visual Studio Code Distilled: Evolved Code Editing for Windows, macOS, and Linux. Berkeley. Apress.

Stackoverflow 2021. Accessed 04.02.2022 https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-integrated-development-environment

Turovsky, B. 2016. Google Blog. Accessed 06.02.2022 https://www.blog.google/products/translate/ten-years-of-google-translate/