Extraction of Image and Data from Engineering Drawings using Python and Tesseract OCR

Saurabh Varughese M. Kovoor

School of Science and Technology,

Sunway University,

Selangor,

Malaysia

20017604@imail.sunway.edu.my

Abstract — This paper presents the development of our program used to extract the desired region of an engineering drawing as well as the data from the tables to be exported systematically to an excel file. Thus, the program was developed using the Python programming language and Google Tesseract for character recognition and detection. Other Python libraries and packages that were used include OpenCV, NumPy, Matplotlib, and Openpyxl. This paper begins with an overview of our project, before progressing to the methodology employed and finally the presentation of its results and discussion.

Keywords— *Tesseract Optical Character Recognition (OCR), OpenCV, Python, image and data extraction, engineering drawings, python to excel export, Openpyxl*

I. INTRODUCTION

The purpose of this project is to develop a python program that can be used to extract numerous information from a set of engineering drawings with multiple layout designs using image processing along with allowing users to identify all the gathered information from those sets of drawings. This includes the drawing, which is the region of interest as well as the table data which will be extracted and saved in a separate excel file.

Tesseract, an open-source Optical Character Recognition (OCR) engine developed by Google was used in this project to aid our developers with processing images as the engine generates accurate outputs and also includes built-in functions that will further assist the processing of images for character recognition and detection.

The report will consist of 3 sections and it will be as follows: libraries and packages included, the methodology and the proposed approach; this section will divulge into how we went about choosing which specific information was relevant to be extracted and identified, the report will also include the results and discussions of the project; this will include the accuracy of the program that was created for this project, along with the outcomes of how the extracted information turned out.

II. USED PACKAGES AND LIBRARIES

A. Google Tesseract OCR, PyTesseract

https://pypi.org/project/pytesseract/

Tesseract OCR is an engine that can be used to identify text in video and images. This will be used to extract the information from images. Pytesseract is the container to use Tesseract OCR with Python and its

functions. To utilise PyTesseract, Tesseract must first be installed on the system before the PyTesseract package can be used.

B. OpenCV

https://docs.opencv.org/4.x/index.html

OpenCV is a library that contains hundreds of computer vision algorithms. This will be used to manipulate the images which can be seen applying filters or manipulating the bits in the pixels directly. The library contains linear and non-linear filtering, colour space conversion, geometric image transformations, histograms, and other features.

C. NumPy

https://numpy.org/doc/

NumPy package is used for scientific and mathematical programming in Python. The library can be used to create multi-dimensional arrays, shape manipulation, and more. This will be used to create masked arrays to highlight certain areas in an image. In Numpy, an array is a table of elements with the same data type. The items are accessible via square brackets and may be initialised with nested Python Lists. Mathematical procedures are also offered to assist in changing the values within an array.

D. Openpyxl

https://openpyxl.readthedocs.io/en/stable/

Openpyxl is a library that can be used to read or write into Excel. This will be used to export text stored in arrays from the table extraction and convert them into worksheets. The library reads data in a logical order from left to right, then repeats the process on the following row from top to bottom until it reaches the end.

E. Other python libraries used

Other libraries from the Python standard library used in the program include DateTime, and difflib.

1. **DateTime**

https://docs.python.org/3/library/datetime.html

DateTime is a module that provides functions to modify time and the date objects.

2. **difflib**

https://docs.python.org/3/library/difflib.html

Difflib is a module that allows the comparison of sequences and returns the similarity ratio.

III. METHODOLOGY AND PROPOSED APPROACH

A. Introduction

Therefore, there are 2 types of data we're aiming to extract from a particular engineering drawing image, which includes a cropped image of the drawing only as an image (png format) and the data of the title block into an excel file (xlsx format). Hence, a key criterion or requirement for this OCR program is to be flexible in operating on different engineering drawings by extracting these information successfully. So, these drawings can differ in terms of the information layout, the positioning and size of the drawing and the data table/title block, font weight, font style, font size, the table's border thickness, and other style elements of the table, inconsistent content or titles available. These possible variations and factors were kept in consideration during our implementation and development of the program.

Overall, our initial goal for this program is to achieve a 90-100% accuracy in extracting and exporting the drawing images, an 80-100% accuracy in extracting and successfully matching the drawing numbers, and an 80-100% accuracy in extracting and correctly matching the other title block data (e.g., amendments table data, project no., unit, drawn by, etc.). In the Results and Discussion section, we'll discuss the method in which we evaluated and calculated this program's accuracy metric.

The program consists of 1 Python (.py) file, extraction.py that acts on a folder of images (Dataset_2021), both of which exist in the same directory. After running the extraction.py file, a Results folder will be created in the same directory, containing folders that hold the cropped images and the excel files with the table data respectively.

B. Contents of extraction.py and Methodology

This is the main driver python file, which performs the drawing and table data extraction. So, it starts by importing the necessary packages/libraries (Pytesseract, NumPy, cv2, os, DateTime, Openpyxl, difflib).

After that, the custom configuration is initialised for the pytesseract configuration, when its methods are called. This particular configuration uses the default OCR engine mode (oem), 3, and page segmentation mode (psm) 6 which assumes the data as a single uniform block of text which we can then process accordingly. Then, a list of words to avoid are initialised, which are words around the drawing that shouldn't be extracted. This will be used to filter the extracted data from the image.

Then, the file contains two user-defined functions that serve a dedicated purpose and will be repeatedly called at different sections of the program. This starts with the formatDate function that converts the extracted date value (string) to the datetime object with the specified format. Hence, ensuring it conforms to the same consistent format "day/month/year" as seen in the original image. Error handling is employed if the string isn't parsed correctly. The second function is the returnSimilarRatio function, which uses the difflib package's method SequenceMatcher to return the similarity ratio (float) between 2 parsed string values, while disregarding any junk elements (None value). So, the higher the returned value, the higher the similarity. This is useful during string comparison purposes in the driver program to conform to a preset level of accuracy.

1. Creating directories to store the results

Next, the program generates the directories to store the program's output using the makedirs method of the os library. Error handling is applied to detect and indicate any OSError or events where the directory can't be created. The exist_ok = True property ensures existing directories with the same name will be overwritten and wouldn't cause program problems. Essentially, the file structure consists of 1 Results directory in the same directory as this code, and underneath that is the respective folders to hold the Drawings and Drawing Data.

The combination of the for loop and if statement repeatedly executes the main code (drawing and table data extraction) for every image ending with .png in the sample_drawings folder.

Part 1 - Extracting the Drawing Image

2. Converting image to grayscale, thresholding, and inversion

Following that, begins the main methodology of this program. After reading (cv2.imread) the image from the sample folder it starts with preprocessing the image to ensure an optimal image extraction. For this, cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) method, converts the read coloured image (with 3 planes) to grayscale form or 1 plane of gray levels. The images number of rows and columns are identified using the .shape method for subsequent processing tasks and mask creations.

The grayscale image is thresholded using adaptive thresholding and simultaneously inverted through the same function. This is to help us easily identify the position of the table borders which will have a high-intensity value.

3. Forming horizontal and vertical lines for mask creation

After that, the image's data is extracted using the pytesseract function, image_to_data(). Then, the output is filtered through the next for loop sequence, which checks and accepts only data which consists of alphabets (isalpha method) and are not part of the wordsToAvoid list or is a date value. Plus, it should meet our specified minimum confidence threshold value, 70 to ensure accuracy. The tesseract output contains values along with the confidence score

corresponding to the likelihood the extracted data is a word. So, if this confidence score is higher than the confidence threshold, the method will accept the data by appending it to the list, given it passes the other subsequent If Else filters as described earlier. Then, the accepted data are appended to the filteredImageData list and used for subsequent processing. Date is the only value containing a non-alphabet character (/), thus it needs to be acknowledged and let to be passed through. This ensures we're extracting only useful data from the image that's either the title or the corresponding value.

The next series of steps involve creating and applying horizontal and vertical line masks. This starts by creating 2 kernels, vertical and horizontal, where the number of rows of the vertical and number of columns of the horizontal is equivalent to a hundredth of the image's width. These are used to detect and locate the position of the vertical and horizontal lines in the image. Then, by performing the morphological operation opening on the thresholded image, the vertical and horizontal kernels are processed to form the lines that mimic the table borders. Then, these horizontal and vertical lines are added together (cv2.add) and dilated (cv2.dilate) using a 2X2 kernel to make the lines slightly thicker.

4. Finding contours in the image

For the next part, we're detecting the contours (cv2.findContours) within the horizontal and vertical line mask, which are the connection of contiguous points of the same colour (in this case having the same high-intensity value). The cv2.CHAIN_APPROX_SIMPLE is used along with this function to approximate and returns only the endpoints that are useful for drawing the contour line. Overall, identifying contours are useful for table object detection. A mask is created with the same size as the image, then using a for loop, for every contour in the image, it will be drawn onto the mask (cv2.drawContours), given the individual contour possesses an area of more than half the image's area. After that the mask is processed through morphology closing to close the empty spaces, using a rectangular kernel or structuring element with a row and column size of a hundredth the image's width. Next, like before, the contours in the mask are detected (cv2.findContours) and an empty final mask is initialised.

5. Drawing contours to create the final mask

After that using a for loop, for every contour in this second set of contours, there are 2 if control statements before drawing the contour to the final mask. First, the individual contour has to have an area larger than half of the image's area. Secondly, the data value from the extracted image data has to exist inside the particular contour. We determine if the point exists in the contour, using the cv2.pointPolygonTest function and if the value is larger than 0, it exists inside.

6. Extracting the table and drawing image using the final mask and bitwise operators

With the correct final mask obtained, we can use it to extract the table from the engineering drawing by applying bitwise_and operation on the grayscale image with the final mask as both contain 1 plane. Similarly, we can extract the drawing only by removing the table using a set of bitwise operators as seen in line 126. Afterwards, the borders of the image are extracted using the same sequence of code as before to draw the contours. Then, these borders are added (cv2.add) to the extracted drawing.

7. Cropping and exporting the drawing image

The next step is to crop only the drawing section of the engineering image. So, first, the image is inverted using the bitwise_not operator to appear as in the original image. Then, using the cv2.findNonZero function we can find the non-zero points, and then cv2.boundingRect to determine the coordinates of the minimum bounding box encapsulating the drawing by returning the X, Y coordinate, width and height. So, this information is used to extract the drawing region of the extractedDrawing image, giving the impression it has been cropped. Then, to ensure the final drawing image appears clearer, a padding of 30 pixels is applied at all sides of the image. Finally, the image is exported (cv2.imwrite) in a png format.

Part 2 - Extracting the Table Data

8. Detecting and Removing the table's horizontal and vertical lines/borders

With the drawing extraction completed, the next task is to extract the table data. So, the subsequent step is to detect and remove the horizontal and vertical lines from the table, making it easier to extract the table data using Tesseract OCR. First, adaptive thresholding is applied to the image of the extracted table, along with inverting its colours or intensity values. Then, like before when extracting the drawing, we'll create horizontal and vertical rectangular structuring where the horizontal has a width and the vertical has a length of 1/160 of the original image width. So, the morphological operation open is applied using these 2 kernels on the thresholded table image. Thus, processing it and mimicking the table lines as in the engineering images. Then, the lines are added (cv2.add) together and used to subtract (cv2.subtract) from the thresholded table image, to isolate the table data without the table lines/borders. The intensity values are inverted to return it to the original colouring and appearance.

9. Extracting and filtering image's table data

Subsequently, the table data text are extracted (pyt.image_to_data). Then the data are filtered using a similar process as before, but with only 1 restriction. The table data should fulfil the confidence threshold level of at least 10, and only then is it stored in the filteredImageData list/array. So, this data consists of the accepted words identified in the image with distinct indices. Next, 2 lists and a variable is initialised for the purpose of exporting table data to the excel file. The first list holds values to be entered into the excel file and the second is to hold values in a particular amendments table row. Then, a 2-dimensional or nested array is initialised signifying 2 fields, the first to hold the main table titles and corresponding values and the second to hold the amendment table titles and corresponding values.

10. Extracting the titles along with their coordinates/location details

After that, 3 arrays are initialised to hold the extracted values, titles and array index of the titles respectively. A letterWidth variable is initialised and calculated based on the extracted image's text's width divided by the number of letters in the particular string along with an added constant. This variable is useful for subsequent comparison purposes to ensure words that are on the same row and are nearby will be combined and considered as one word/phrase. The first nested for loop is employed to extract the titles and their coordinates. It begins by looping through the filtered word list, retrieving its coordinates, capitalising it, checking if it's longer than 1 character. Then, it checks if the nearest or next word is in the same y-axis (assuming on the same row), both words are combined to form one and are nearby on that row (using the letterWidth variable). Another nested for loop is used to compare the similarity of the extracted title with the actual title as initialised in the field array. So, if the similarity between both words is more than our specified threshold ratio of 0.8, the extracted title and its index is accepted and appended to the newly initialised array for titles and removed from the 2-dimensional field list. Overall, this is useful for different types of engineering drawings that don't utilise the same format or spelling for a particular title, so the original title is extracted.

11. Extracting the content and combining disjointed words to form 1 phrase/word

The second sequence of for loops is to combine content/words that should be together but mistakenly identified/extracted as 2 separate words. An example of this is in the Drawing Title or Number which often contains multiple parts but should be treated and identified as a single word or phrase. Therefore, for every word in the OCR extracted list, the program checks if the next word is on the same y-axis row, then it checks if the words are close together in the x-axis. It does this by finding the difference between the 2 words and if it is lesser than the letterWidth value from before, the words are combined and it along with its coordinate are appended to the extracted words list.

12. Using the title to extract the closest content and assign it to its corresponding value

The third sequence of for loops uses the location/coordinates of the titles to extract the corresponding values which are located nearest to the title. First, four variables required for the data

extraction are initialised (contentIndex, skippedIndex, extractedWord, nearestContent constant). Then, for every item in the extracted word list, it first obtains the coordinates of the current title and current extracted word. It uses the Pythagoras theorem formula to calculate the distance between the title and corresponding value, given it's nearby on the x and y-axis. Then, if the distance is less than the nearest_content constant, it's accepted. After that, there's an if checking statement to ensure the titles (DRAWING NO.:, DRAWING Number:, Project No:) that are extracted fulfil a similarity ratio with the predefined titles in the fields array of more than 0.8. It also performs a checking to ensure the extracted word is not blank. Then once the checks are passed, the combination of extracted titles and their corresponding values are linked, accepted and appended to the excelInput array, which is used for adding the values into the excel file.

13. Extracting the Amendments table data

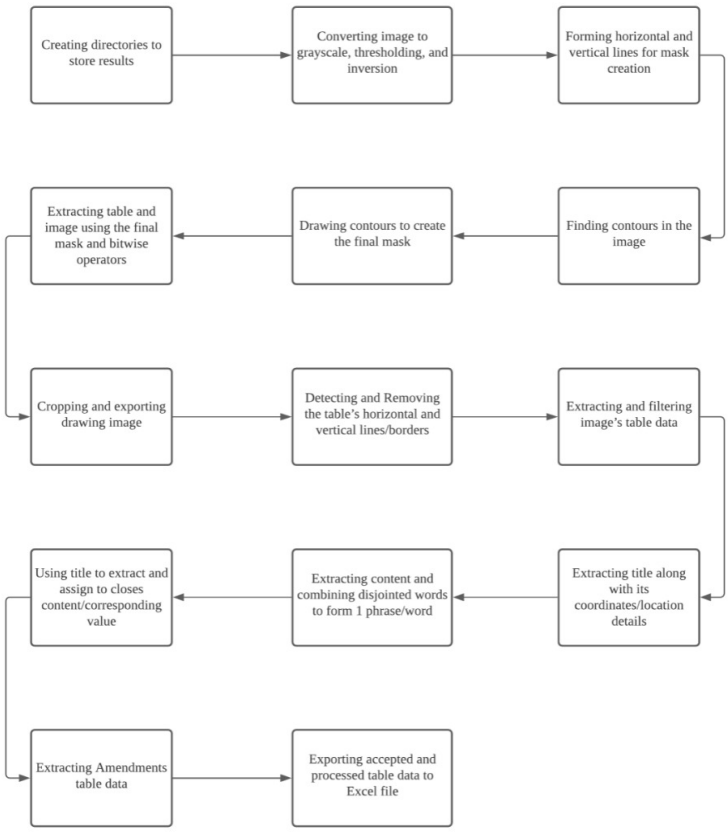
The final for loop is to process the remaining extracted words, which are from the Amendments table. So, it starts by extracting the particular word and its coordinates. The first word (AMENDMENTS) is automatically appended to the amendmentRow list and subsequently, the excelInput array as this is the only word in its particular row. After that, the next loop extracts the titles that are in the same row or y-axis. Once all the data in a particular row is extracted it's stored in a single amendmentRow list which is then appended to the excelInput array. This ensures that all the inputs in a particular row in the amendments table appears identically in the same row in the excel file as well. Then, the subsequent loops extract and append the corresponding values of the titles to the amendmentRow list, which will be appended to the excelInput array. Afterwards, the amendmentRow is emptied allowing for the new row in the amendments table to be extracted.

14. Exporting accepted and processed table data to an Excel file

Finally, with all the extracted data completely processed, the extracted and matched contents of the table data in the engineering drawings will be added to an excel file. For this, the program starts by initialising the workbook, then retrieving the active worksheet. Next, 2 for loops are used signifying the row (outer for loop) and column (inner for loop). Then, the title and corresponding value is parsed into the excel sheet. Finally, the workbook or excel sheet is saved (.save) with a filename based on the original image's name.

C. Flowchart of Proposed Approach

The following flowchart summarizes our proposed approach and the series of general steps to extract the drawings and table data from the given set of engineering drawings effectively and accurately. Thus, this diagram was prepared beforehand to provide a conceptual overview of the steps and processes that need to be taken or performed:



IV. RESULTS AND DISCUSSION

The following section contains the obtained or calculated extraction accuracy results. This includes the accuracy of the output of the engineering drawings (*"filename"_drawing.png*) and the output of the table's content (*"filename"_drawingInfo.xlsx*).

A. Accuracy Tables (For Table Data)

Title	Accuracy
Drawing Number	**17/20 = 85%**
Drawing Title / Title	**17/20**
Company Name / Company	**2/2**
Contractor	**17/18**
Drawn / Drawn By	**18/20**
Checked / Checked By	**18/20**
Approved / Approved By	**17/20**
Unit	**20/20**
Status / STS	**19/20**
Page	**10/19**
Project No	**10/18**
Lang	**16/17**
CAD No	**13/16**
Font	**0/2**
Overall Percentage of Main Table Titles (excl. Drawing Number)	**78.69%**

Title	Accuracy
REV / Issue	**19/20**
Date	**12/20**
Changes (only 4, 5)	**2/2**
By	**16/20**
CKD (only 4, 5)	**2/2**
Overall Percentage of Amendment Table Titles	**87%**
Overall Percentage of All Table Titles (Excluding Drawing Number)	**(78.69 + 87)/2 = 82.85%**

B. Drawing Extraction Accuracy

In terms of drawing extraction, it is expected the program should be able to successfully identify the location of the drawing, crop to only include that particular region, and then export it as a "filename"_drawing.png file. So, extracted drawing image shouldn't include any outside elements, such as the tables or its borders either. If these criteria are fulfilled, then the drawing extraction for that particular engineering image is marked successful. Therefore, in this department, our program achieves a **100% accuracy rate** when tested on all 20 sample images.

C. Drawing Number Extraction Accuracy

In terms of the drawing number extraction, along with the other table data we intend to extract, we prioritise extracting and matching the DRAWING NO. field. So, we expect the program to extract and match both the DRAWING NO. field and its corresponding value so that it appears in the same row in the final excel file. It should maintain the same punctuation and appearance as seen in the engineering image. If these criteria are fulfilled, then the drawing number extraction for that particular engineering drawing is marked successful. Therefore, we found that our program achieves an **85% accuracy** in extracting and matching this drawing number field when tested on all 20 sample images. Only on images 5,10, and 17 were the drawing number value either not matched together with the field title (5, 10) or not extracted at all (17).

D. Other Title Blocks Extraction Accuracy

Apart from the drawing number, there are other fields in the main table that need to be extracted as well as the fields in the Amendments table. Thus, similar to before, the goal is to extract and match the fields with

the corresponding values, so that it appears in the same row in the final output excel file. For the Amendments table, the values should be in the same column as its corresponding title/field. All the spelling and punctuation of these values should be identical to that as seen in the engineering image. So, if these criteria are fulfilled, then the title block extraction for that particular engineering drawing is marked successful. Overall, we calculated the success rate in extracting all other title blocks in the main and amendments table to be **82.85%**. The errors we found during extraction include instances where the value and title weren't matched or matched to different titles, incorrect value spelling or content, or in some cases the value wasn't extracted.

E. Effects on Accuracy

This subsection discusses the justifications for the extraction inaccuracy observed in several instances. Based on analysis of the samples provided and comparison with the information extracted, the following are three possible reasons we found to affect the extraction accuracy:

Font Type

From our observations and analysis of the program outputs, mainly the table data extraction, we found that font changes have an impact on the accuracy of the words extracted. Every image in the 20 sample set uses the same Arial sans-serif font type, except for image 16 which uses a serif courier new. Owing to its slimmer style it is easier and more accurately identified by the Tesseract OCR compared to the other images' Arial font. Hence, most of the fields for that image were successfully extracted, except for font. However, it can get affected by erosion, which will create gaps within the lettering and cause an unsuccessful extraction. Thus, we didn't employ any erosion on the table data. However, this should be kept in mind, when processing or refining the image using erosion in future. Overall, we can conclude that slimmer font types are more legible and accurately extracted by Tesseract OCR.

Layout Design

From looking at the sample engineering images, we notice groups of images having the same layout design or the positioning of the tables and their data/fields. So, some layout designs had more errors during title block extraction than others. From our observation, when all the table data are positioned in one straight column at the left or right side (e.g. images 7, 8, 20), was when extraction of data is often successful. Thus, when the tables are disjointed (e.g. images 14, 17, 18) it involves errors with extracting and matching data. We believe this is because the fields are arranged more sequentially row-by-row when arranged in the same column, allowing for easier extraction and processing by our program than in disjointed tables. So, the values would not be affected by values from other titles/fields.

Additionally, some data such as drawing number, the values are separated by lines. This can further distance the values apart and often causes problems when identifying it as a single entry/word. This is the case for image 17 where the drawing number couldn't be extracted successfully as it was identified as separate segments. Thus, it's more optimal for there not to be any separating lines in the values.

Other Difficulties in Information Extraction

There are particular fields that our program experienced many difficulties in extracting. Firstly, when extracting the date field in the Amendments table or the page field in the main table, it often mistakes the "/" symbol for 1 or another symbol such as "V". Thus, the OCR mistakenly identifies it for other characters that look similar. Hence, why Page and Date have a relatively low accuracy of 52% and 60% respectively. To rectify this, we can try preprocessing that high-risk section of the image with erosion to make it slightly thinner and distanced from the other characters, so making it easier for the OCR to recognise it as "/".