An Extensible System for Optical Character Recognition of Maintenance Documents

John A. Labarga[1], Amardeep Singh[2], and Vera Zaychik Moffitt, Ph.D.[3]

[1,2,3] *Lockheed Martin, Shelton, CT, 06484, USA*
john.a.labarga@lmco.com
amardeep.singh@lmco.com
vera.zaychik@lmco.com

ABSTRACT

In the course of maintenance and operations, equipment operators and manufacturers frequently generate large volumes of paper documents. This is particularly the case in maintaining legacy systems, and when external factors (e.g. security concerns, environment, training procedures) make it infeasible to record data in a computer system at the work site. These paper documents may contain essential information about assets, such as their cost, performance, and reliability. To implement analytics or automated monitoring, these documents must later be converted to digital copies, which can be ingested into a database. This paper describes a flexible system for converting tabular paper forms into digital documents through Optical Character Recognition (OCR), utilizing open source tools and packages. The resulting application offers a template-based configuration interface, which allows it to handle a variety of document types. This system also allows for the incorporation of business rules and processes that deliver high fidelity digital copies.

1. INTRODUCTION

In many settings, maintenance documents are available predominantly or exclusively in a paper format. These documents may contain critical data about the status, performance, and configuration of an aircraft. As a result, aircraft operators and manufacturers prefer to have this data available in a digital format. This generally requires scanning a document to produce an image, using OCR to produce a digital copy of the text, in an appropriate format for a database.

While a variety of Commercial Off-The-Shelf (COTS) OCR solutions exist, they are typically designed for free-text documents (e.g. letters, book pages) and simple forms. When processing complex forms containing irregularly-shaped fields and many sections, these applications often deliver improperly-formatted, low-fidelity output. A custom solution, incorporating domain expertise and business rules, can deliver higher performance in this setting.

This paper describes an extensible framework for building a tool to perform OCR on custom forms, with a focus on applying this framework to maintenance documents. This framework consists of open source software, and can be readily adapted to new applications using the techniques described.

The framework consists of two basic steps: decomposing the form into a series of elementary sections, and performing OCR on these sections before reassembling them into a digital document. The framework is sufficiently general that, after building a tool from it, the user can directly convert an image of a paper form into a spreadsheet or SQL table.

To achieve high-fidelity OCR output, the document must be broken down into small chunks of text. In the case of a paper spreadsheet, the natural segmentation approach is to split the document into its cells. In order to do this, the authors use the OpenCV (Bradski, G., 2000) python package, which provides a variety of image processing and computer vision utilities. This package is used to extract cells from the form, resulting in a series of smaller images consisting mostly of unstructured text. Section 4 describes strategies for document image segmentation to achieve optimal OCR performance.

These image segments are then passed through an OCR program to extract the text content. For this purpose, the authors use Google's Tesseract OCR engine. Tesseract is a customizable, retrainable OCR solution, which achieves high fidelity when used for unstructured text OCR. While it is packaged with the ability to read standard fonts with high accuracy, Tesseract can also be modified to perform OCR on custom fonts, as appear in typewritten documents and internal forms. Section 5 describes how to modify Tesseract's character recognition models to adapt it for custom fonts. This paper also shows how to provide the OCR engine with regular expressions, which boosts OCR performance on part numbers, dates, etc. After OCR is complete, further improvements are made by applying business rules. If a form entry commonly has a specific format (e.g. a telephone number), then a post-OCR processing rule might look for text which is in approximately the same format as a telephone number. This approximate match for a telephone number is then formatted appropriately.

Section 7 describes how to reassemble the OCR output into a digital document. This discussion includes strategies for handling errors in the extraction and OCR processes. Finally, Section 8 discusses the evaluation framework and performance the system achieves on a representative dataset.

2. MOTIVATION

Comprehensive and accurate data is essential to the monitoring and maintenance of industrial systems and assets. Increasing the amount of data available about an asset improves an organization's overall understanding of its own operations, which enables better decision-making. In addition to the benefits of collecting comprehensive data, the expense and duration of analysis can be reduced by storing data in an accessible format, e.g. in an organization's database rather than as physical records.

For example, an airline may seek increased aircraft availability. After some analysis, it may be found that the primary cause of unavailability is schedule overruns in engine maintenance, caused by underestimation of maintenance scope. The airline's goal is to better anticipate the scope of maintenance, to have components and technicians in place to complete the work on time. This effort is more likely to succeed if detailed maintenance logs and component invoices are available, than if only the frequency of engine maintenance has been tracked.

Many organizations have significant amounts of historical data stored in an analog format, in support of long-standing process management and compliance functions. Among these are free-text documents, physical record books, and custom forms. While they preserve data, these formats are not conducive to computerized data mining or analysis. Each non-digitized physical record represents a gap in the organization's analytic capacity, as information that is unavailable for statistical analysis and visualization.

A common approach to digitizing such documents is manual transcription, in which the document's overall format is reproduced in a digital form, and its contents are entered by hand. This process is costly, and may introduce errors into the organization's databases. Manual transcription requires monotonous, fatiguing work that is prone to mistakes. This is in contrast to a software approach, where the marginal cost of digitizing a document is low, and the performance is consistent.

In view of these issues, some organizations have begun to offer generic document OCR solutions.[1] While typically oriented toward free-text documents, the ability to handle common or simple forms is sometimes offered with this software. As a result, an off-the-shelf solution is sufficient for some applications.

While commercial OCR solutions have benefited from decades of research in computer vision, they still have performance and quality issues when non-text marks are in close proximity to the desired text. This problem is amplified for documents with grids and other structural features. Feeding an entire log card image into a COTS OCR solution often results in output of inconsistent quality. The authors tested a COTS OCR solution against a sample of forms, and found the performance to be poor: the software was able to OCR and reconstruct only 45.74% of the cells in the forms. Particular problems included vital text being misaligned and populated in the wrong section, grid edges assumed to be characters resulting in spurious text, and entire cells and sections missing due to image quality issues.

In order to address these problems, a framework is needed that can decompose a document into a series of simple sections to which available tools can be applied. This method would remove most edges that can cause low fidelity in the OCR output, but also allow for independent control of each segmented image during later processing. The format of the document and features of the data contained could be incorporated into the processing of these images, yielding an extensible OCR system. Having control of this workflow can also increase the fidelity of the OCR process.

In this application, maintenance documents have numerous data types depending on cell and section, ranging from dates and part numbers to free text. COTS software did not allow for any inputs such as regular expressions to fine-tune the output. Developing an independent OCR pipeline is not only advantageous for pre-processing inputs such as regular expressions, but also for incorporating post-processing algorithms based on cell type to further improve the output. Since the software was developed in-house, it can easily be adapted to new forms and documents. Overall the advantages of developing an in-house solution outweighed the low-cost but highly inconsistent output from a COTS platform.

3. DATA PREPARATION

Extracting and analyzing the rich data in maintenance documents can have numerous impacts for a product, including identifying insights that allow for maintenance reduction, cost savings, and even improving product life cycle. The method by which a maintenance document is created, maintained, and tracked can vary drastically across product lines and industries. In the case of helicopters, maintenance doc-

[1]Examples include software produced by Nuance, Dropbox, and Adobe.

uments, also known as log cards, stay with the part until the end of the part's life. Mechanics and aircraft operators do not always have access to a computer while in the field and as a result will update these documents with a portable typewriter or hand-written text. Since the log card is always in a physical medium, it can accumulate damage over the lifetime of a part, further reducing legibility. A typical log card goes through numerous iterations of scanning, printing, and manual updating by typewriter or by hand. This process often results in the log card containing erroneous marks, gaps in lines, faded text, smudges, and other defects. Before the valuable data in a log card can be converted into digital format for analysis, it is paramount that the physical document be scanned at a high image quality.

A low quality scan will create more errors downstream during the image segmentation and OCR phases. Through numerous tests of the OCR performance, a minimum scan quality of 300 dots-per-inch (DPI) was identified to be ideal. When comparing a 200 DPI and 300 DPI scan the difference in quality may not be immediately visible to the human eye; however, for a computer reading the image at the pixel level, a 200 DPI scan results in a degradation of image quality which can reduce OCR performance by more than 50 percent. Furthermore, the file format the log card scan is saved as is also important. Since the image quality is vital to the process, it is important to save the scan in a lossless compression file format such as Portable Network Graphics (PNG), as opposed to a lossy compressed file format such as Joint Photographic Experts Group (JPEG).

A product can contain hundreds or even thousands of parts with varying maintenance content and document formats. A complex and sophisticated product such as a helicopter contains numerous systems and subsystems that may have different maintenance document schemas. A schema covering components from one system may contain specific sections of data that are not applicable or shown on another schema with different parts. The authors found that implementing one representation for all variations would be impractical and result in a highly fragile and error-prone system.

A solution to this problem is to identify the number of document schemas that represent the target population and develop a template for each schema. Each maintenance document is matched with its corresponding template, and specific business rules and methods encoded in this template are applied during the image segmentation, optical character recognition, and reconstruction phases. The template approach yields an extensible and reusable framework for performing OCR on forms. This software can be extended to a new document by producing a corresponding template, and incorporating it into the system.

4. IMAGE EXTRACTION

Each log card is a combination of sections, where each section is approximately a table grid with a title, column headings, and one or more data rows. The goal of image segmentation is to locate each content cell and save it as a separate file with the correct section and cell number identified. This allows other processing components downstream to take advantage of the cell context information to improve overall accuracy, as we discuss in Section 5. A powerful and robust computer vision library available in Python called OpenCV was utilized for developing a solution to segment the image into its sub-cells.

Image segmentation consists of the following steps:

1. Extract each page of the log card. The following steps are performed on each page.
2. Locate the overall input grid using cropping and rotation as necessary.
3. Segment the image into card sections that have different semantic meaning. The following steps are performed on each section.
4. Overlay the grid of row and column lines to find every input cell. For each cell in order, crop lines and save as a separate image, if not blank.

We discuss each step in turn.

4.1. Page Extraction

The scanned log cards are provided as multi-page PDF files. We use GhostScript (Artifex Software, 2018), an open-source library, in batch mode to extract an image file for each page. The rest of the processing is performed using various image functions of OpenCV. The quality of the saved image is variable. Maintenance documents can travel with a part for years and as a result the physical document often contains smudges, hole-punches, spurious characters, erroneous lines, etc. This document abuse becomes even more pronounced when the document is scanned and converted into an image.

4.2. Cropping and rotation

The goal of this step is to orient the main grid such that only the card itself remains and the horizontal row lines are parallel to the bottom and top of the grid, while the vertical column lines are parallel to the left and right of the grid. The alignment of the grid in this fashion simplifies all the subsequent steps of finding and extracting card sections. The simplest approach is to detect the largest external contour. However, in practice the image may have line gaps due to hole punches, spurious text outside of the grid but close to it, handwritten lines that extrude from the grid, and other issues that prevent reliable correct contour detection. We need to inscribe the largest quadrilateral into the largest contour hull. In the absence of an efficient algorithm for doing so, we locate the four corners and connect them. The warp perspective function then can take the four corners and crop/rotate accordingly. Figure 1 shows an example log card with the largest outer contour and the detected grid border.

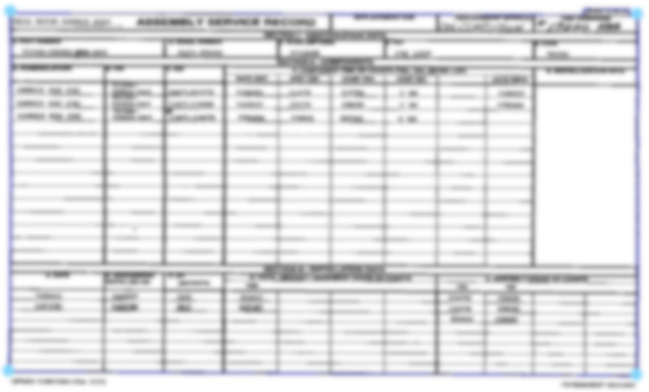


Figure 1. The log card is slightly askew and some of the text is outside the grid but close enough to be part of the detected contour. The blue line is the largest external contour that OpenCV detected. The light blue dots indicate the detected box corners. The image is blurred to obscure sensitive information.

4.3. Section extraction

Each log card page consists of two or more sections that are essentially tables to capture different types of information. Since each section follows a consistent layout of a title followed by column headings and one or more content rows, the most reliable way to segment the page into sections is to detect each section title cell that spans the page width, which can be done by detecting contours in OpenCV. The space between the section titles or between the title and the bottom of the page is a single section. Sections are numbered sequentially. A similar process is used to eliminate the column headings. Figure 2 shows an example section with the heading candidates identified.

4.4. Cell extraction

To detect the row and column lines forming the cell grid, the Canny edge detector (Canny, 1986) works well. The detected lines do not span the height or width of the image because of line breaks from fading and other defects from the scanning process. Each detected line can be used as a candidate in a voting process (Matas, Galambos, & Kittler, 2000). The voting is important because of occasional hand-written word cross-outs and other noise, as can be seen in Figure 3. The results of the voting process, consensus lines, form the grid of individual cells. Each cell in order from left to right and top to bottom is mapped to its label based on the template. The template, read in from a file, specifies which cells to skip

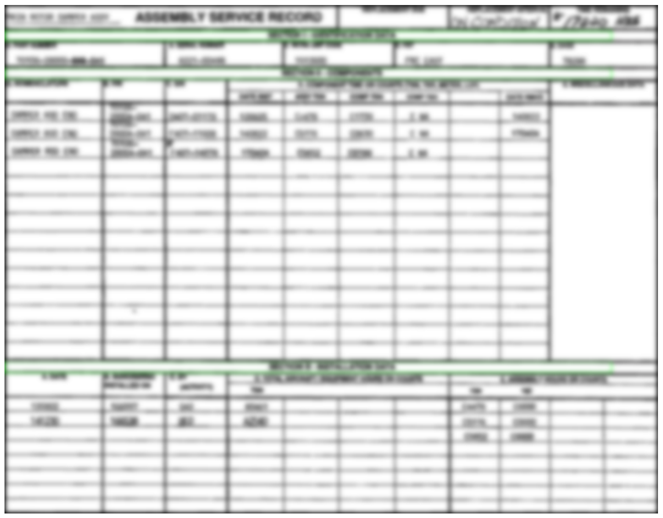


Figure 2. A single page of the log card with the section headings identified with green rectangles. The cells span the image width and separate one section from another. The image is blurred to obscure sensitive information.

and is a mapping from raw section and cell numbers to output section and cell numbers. In Figure 3 one highlighted cell is from section 2 and is 16th in a sequence, but is saved as cell 2-15. If a cell is blank – image mean intensity is above the threshold – then it is not saved.

The cells, once saved, go through to the next stage: OCR.

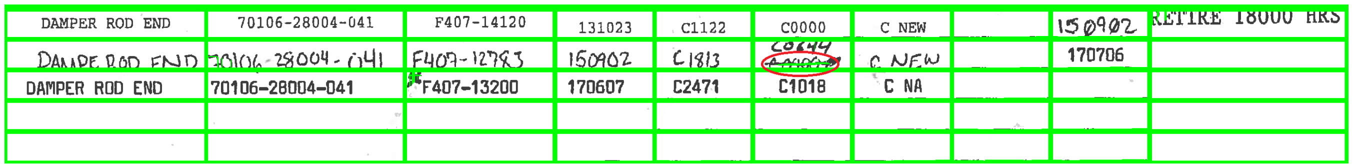
5. OPTICAL CHARACTER RECOGNITION

Optical Character Recognition is the process of converting an image of text into a digital representation of that text. It is an essential component of a document digitization pipeline, since the alternative is time-consuming manual transcription. The result of OCR is a digital representation of the text in an image, which can be further processed or loaded into a database.

Regardless of the particulars of an implementation, OCR fidelity is generally a function of the complexity of an image. For a given OCR program, higher fidelity can be achieved on a simple image (e.g. a line of text from a book), than on a complex image (e.g. an image of a spreadsheet). This is the motivation for the cell extraction procedure described above; the contents of an individual cell can be converted with higher accuracy than a spreadsheet, or section thereof.

5.1. Potential Strategies

Since OCR is necessary for efficient document digitization, various approaches exist to implement it. In general, the two basic strategies are to build an OCR solution from scratch, or to adapt a prepared solution to the documents that are relevant to the application.



DAMPER ROD END	70106-28004-041	F407-14120	131023	C1122	C0000	C NEW		150902	RETIRE 18000 HRS
DAMPER ROD END	70106-28004-041	F407-12783	150902	C1813	C0644 ~~[illegible]~~	C NEW		170706	
DAMPER ROD END	70106-28004-041	F407-13200	170607	C2471	C1018	C NA			

Figure 3. Line detection is used to overlay the grid to extract the cells. However, spurious lines must be ignored, like the one encircled.

One approach is to build a novel OCR program, using a machine learning algorithm. The MNIST dataset, a common benchmarking dataset for computer vision algorithms, consists of a large number of labeled handwritten numerical digits from the set $\{0, ..., 9\}$. This dataset can be used to construct a novel OCR solution that can convert images of handwritten individual digits to digital representations of numbers. Similar datasets are available for handwritten characters and for typed text, to train a model to perform OCR on handwritten and typed words and sentences.

An advantage of training a custom OCR solution is the ease of tuning the model to domain-specific fonts or characters. If the text to which the model will be applied is different from that on which it was built, the model will tend to perform poorly. This is a weakness of off-the-shelf OCR solutions, which tend to be trained on extracts from books or newspapers; these extracts might bear little resemblance to the text on the documents for a specific application. A custom solution can simply take examples of this font, and add it to the model's training set.

However, constructing a character classifier from scratch is not an efficient solution to this problem. One issue that arises is that although the character classification effort may achieve high accuracy, an effective OCR solution must also learn to decompose sentences into words, and words into characters. This is a non-trivial problem, requiring additional software and modeling.

Additionally, a custom character classifier is unlikely to outperform tuned models constructed by experts. Publicly-available OCR models, constructed by researchers in computer vision, can achieve character-level accuracies of 99% or higher. A novel solution is unlikely to exceed such performance.

Finally, the amount of training data needed to construct a custom OCR solution is an obstacle to this approach. The MNIST dataset consists of 60,000 examples, or about 6,000 examples per digit. While typed text is less challenging to recognize than handwritten text, an effective OCR solution would likely still require hundreds of examples per glyph to achieve good performance. To train a custom OCR solution would require an upfront investment in producing potentially tens of thousands of training examples.

The alternative to building a custom model is to use an off-the-shelf OCR tool. As previously discussed, a complete piece of OCR software is often not ideal for this purpose, since it may perform poorly on detailed forms. Thus the strategy is to extract the individual fields and apply an OCR tool to each. A fast, scriptable tool (i.e. not a GUI tool) is needed to do this, since a form may contain hundreds of these extracts. Ideally this tool will also be trainable, to accommodate unusual fonts and characters.

5.2. Tesseract

Tesseract (Google Inc., 2017) is an open-source OCR tool that offers a robust training set and the ability to add custom training examples. Originally developed at HP in the 1980s, the tool is now maintained and developed at Google. Tesseract is commonly used as a back-end OCR solution, and has APIs available in several languages, including C++ and Python. Comparisons have shown Tesseract to be faster and more accurate than competing back-end OCR solutions (Patel, Patel, & Patel, 2012).

Tesseract can be used without any modifications, but also offers the ability to add domain-specific training examples. This increases its performance on custom fonts and character sets; untrainable off-the-shelf tools will tend to underperform in this case. This retraining process is described in the documentation, and uses other open-source tools to produce the necessary training data.

In addition to providing high-fidelity OCR at the character level (Smith, 2007), Tesseract incorporates an English dictionary to improve its performance on common words. The user can add custom words to this dictionary; this is an important feature, since many terms that are common in support documents are rare in books and newspapers (e.g. part names, malfunction terminology). Tesseract also allows the user to supply common patterns in the data, i.e. regular expressions which commonly appear in their documents. This is useful for improving performance on serial numbers, physical measurements, and other fields with entries that typically conform to some regular format.

Tesseract is trained on typed text, and generally does not perform well on handwritten text. If handwritten text is clear and consistent, it may be possible to introduce examples as would be done with a font. However, handwritten text OCR is an area of active research, and emerging solutions are likely more appropriate than Tesseract.

5.3. Adapting Tesseract to Custom Fonts

The image set used to train the base distribution of Tesseract consists of extracts from books and newspapers. As a result, the software's performance is high on fonts that are common in these sources (e.g. Times New Roman), but may be considerably lower on typewriter fonts and fonts particular to word processing software.

If the OCR fidelity on custom fonts is unacceptable, it is possible to add extra training data (i.e. labeled example of the custom font) to Tesseract and to then retrain the character recognition models. This process involves gathering and labeling a sample of images containing text written in the custom font. The necessary size of this sample varies with the complexity of the glyphs, as well as the number of glyphs. For example, a complex font such as Blackletter, or a language with a large character set such as Chinese, will require a larger image sample than a typewriter font.

Once a sufficient sample has been collected, the glyphs must be labeled with their character values. Open source software is available for expediting this task, such as JTessBoxEditor. This software automatically identifies glyphs in the sample images, and allows the user to input the appropriate value. In the event that glyphs are mis-identified, the user can modify the "bounding boxes" that isolate and individual character.

This process yields a set of files that can be fed into Tesseract, which will retrain its character recognition models to recognize the supplied examples as a discrete font. This will improve the OCR fidelity when using Tesseract on documents containing this font.

5.4. Regular Expressions and Custom Words

Maintenance forms often contain data that conforms to some pattern, or contains some jargon that is uncommon in general English. In these cases, it can be helpful to register such patterns or jargon with the OCR engine, to increase the likelihood that these elements will be recognized correctly.

Tesseract allows the user to supply two text files, containing custom patterns and custom words. These files are referenced as OCR is taking place. For example, if the data commonly contains phone numbers in the format XXX-XXXX, the corresponding regular expression is `\d{3}[-]\d{4}`, which can be added to the user patterns file. Similarly, the word `widget` might appear frequently in the data, but rarely in general English. This word should be included as a user word to increase the likelihood that it will be recognized correctly.

5.5. Using Tesseract in an OCR Application

The Tesseract API provides a variety of endpoints for extracting digital text from images. These can be arranged to increase the accuracy OCR output, while conforming to business rules and integrating fully with the rest of the document processing application. While supplying domain-specific regular expressions and words generally improves the quality of the OCR, the output often still contains some errors, which can be corrected by accessing these API endpoints.

The principal API endpoint, GetUTF8Text, returns the OCR result for an image. Tesseract can also be run as a command line program; UTF8Text returns the same result as if the command line utility were being used. The result is a plaintext string, which can be passed to the document reconstruction utility.

Other endpoints offer more verbose output, which can be used to improve the OCR fidelity. Among these is MapWordConfidences, which returns a series of (Word, Confidence) pairs. Each discrete word found in an image by Tesseract receives a confidence score. This utility is useful for filtering out low-confidence results, while retaining high-confidence results, which are likely correct. This solves a common problem in recognizing the text in a cell: the OCR utility will sometimes recognize non-word entities, and try to assign characters to them. For example, the cells in a form may have stray marks, hand-drawn notations, scanning artifacts, etc. These will appear in the MapWordConfidences output as low-confidence pairs, which can be filtered out by setting a threshold on the confidence.

At this stage, a custom dictionary can be imported and compared against the individuals words identified by OCR. If a word identified by OCR has low confidence and is similar to a dictionary word (e.g. `BEARING` mistakenly read as `BEAR1NG`), a substitution can be made.

Another useful endpoint is GetChoiceIterator, which can be used to enumerate the possible choices for a character. For example, it is common for 0 to be errantly read as O, B as 8, etc. The character iterator can be used to construct different possible ways to read a cell, by iterating through the choices for a character and assembling the corresponding OCR candidates. This is useful when a cell must conform to a regular expression. If the initial OCR result does not fit this regular expression, the choices can be iterated through to find a combination of candidates that does. This supplements the aforementioned pattern dictionary, to guarantee that the OCR of a cell conforms to a given type.

As an example, consider a situation where an image truly contains the text `B1234`, but is erroneously recognized as `81234`. Extracting the candidates for each glyph may yield $\{\{8,B\},\{1,I\},\{2\},\{3\},\{4,A\}\}$. The alternative recognitions assembled from this candidate set are

{81234,B1234,...,BI23A}. If the content of this cell is known to be one letter followed by four numbers, then only one of these alternative recognitions can be correct.

In practice, these regular expressions are read from a template describing the overall structure of the document. Typically only a subset of the document contains data which conforms to a type. A practical consideration for the procedure described above is that for long regular expressions, or glyphs for which there are many candidates, the number of alternative ways to recognize the cell may be large. The number of recognition alternatives generated by this procedure is

$$\mathbf{S} = \prod_{i} |C_i| \tag{1}$$

where C_i is the set of candidates for the ith glyph, and $|C_i|$ is the size of that set. $\mathbf{S}$ can be calculated before generating the recognition alternatives. Based on the available computing resources, a limit can be set on $\mathbf{S}$, above which the described procedure will not be performed.

When the OCR process for a given cell is complete, the digital text is returned to the application. The data for each cell is then reassembled into a digital representation of the document.

6. DATA MAPPING

An extensible system adapts and scales according to the provided input, in this case the log card being processed. As a result of log cards having various formats, corresponding parameters and business rules are applied based on its schema. A template file containing all inputs needed to efficiently process the log card is vital to the system. This template file contains inputs that are used downstream in the image segmentation and OCR processes. During the image segmentation process this template is referenced to correctly label each image segment. Like a typical table, a log card also contains cells with various data types depending on the column it is located in. To improve OCR performance the template highlights what regular expressions need to be applied based on the data type of each cell.

7. DOCUMENT RECONSTRUCTION

The final process of the system is to intelligently aggregate all the segments of data that were created during the optical character recognition process and recreate the original log card. The template file is the root source that helps achieve this since it is utilized in the naming convention for the image segmentation process. Each file output for the prior phases were named according to the template being processed, section number the file belongs to, and the cell number in that section. Table 1 shows an example of this naming convention.

Table 1. This naming convention allows for each output file to be mapped to its corresponding location during the reconstruction process.

Filename	Template01-2.11.txt
Template Name	Template01
Section Number	2
Cell Number	11
File Extension	.txt

A preliminary method of outputting the OCR data into a spreadsheet was created using a Pandas data frame structure since it closely resembled a table with column and rows. A data frame was created for each section of the template with column headings and valid cell number data extracted from the initial template file. As the system read each OCR text file it utilized the file name to locate the matching data frame and cell number to transcribe the OCR data. Once an entire log card was processed the data frames were output into a spreadsheet. A more robust method of exporting the data into a database is still under development. The file naming convention will once again play a key role in order to map each data point to the correct field in a SQL table.

8. PERFORMANCE

Two key categories needed to be tracked for accuracy to validate overall system: image segmentation and optical character recognition. A truth data set for each template needed to be established to measure accuracy across the system. Due to the nature of the problem of converting physical data into digital data, developing the truth data required manual effort and could not be automatically generated. Since the pool of maintenance documents would grow by 70-100 each month it was not feasible to generate truth data for each document each month. To add to this issue each log card scanned did not comprise of the same image quality. To combat this problem, the population at the time of 300 log cards, were analyzed and divided into three complexity levels of low, medium, and high. The complexity level was based on the following criteria: Typed Text, Legible Handwritten Text, Illegible Handwritten text, Faded Text, Erroneous Marks, Background Noise, and Image Orientation. A log card that was considered a low complexity level, i.e. system will require minor post processing to improve quality of image extraction and OCR, contained only typed text, and few erroneous marks. Log cards classified as medium complexity contained a hybrid case of typed text, legible handwritten text, faded text, and minor background noise. Lastly log cards classified as high complexity contained little typed text, illegible handwritten text, significant erroneous marks, faded texts, background noise and orientation issues. Table 2 shows the complexity breakdown of the 300 evaluated log cards.

Truth data for the image extraction and optical character recognition phases was generated based on the above com-

Table 2. Log card complexity breakdown.

Low	Medium	High
56.7%	23.8%	19.5%

plexity breakdown to ensure majority of the cases were captured. This also allowed for a more true representation of the overall system performance.

The image segmentation phase has the greatest impact downstream, directly affecting the optical character recognition and reconstruction phases. The more errors this phase has the more errors occur during the OCR and reconstruction phases since the output from the image segmentation phase is the input into the subsequent phases. Common errors discovered included improper segmentation resulting in combined segments, missing segments or even cut-off segments. To minimize errors downstream this phase required rigorous testing and also had the largest truth data set of 100 log cards equating to over 7000 image extracts. Accuracy for this phase was measured at the document level. For example, if a maintenance document had 100 valid image segments then the test case needed to output all 100 image segments to pass testing. If the test case did not output the same cells as the truth data for that document it failed and needed to be re-evaluated. Current accuracy of the image segmentation phase for the first template is at 88% and at 90% for the second template. Subsequent templates are still under development and have not been tested.

The optical character recognition phase had a smaller data set of 50 log cards resulting in a population of roughly 3000 files. OCR accuracy was evaluated at the cell level. When a log card was evaluated for accuracy each cell was compared to its corresponding valid cell in the truth data set. A test cell having a 100% match with the truth data cell resulted in a pass. This process was applied across an entire log card and the results were aggregated to get the OCR accuracy for that log card. A system level accuracy score was obtained by calculating the mean of all log card scores. Current system level accuracy of the optical character recognition phase for the first template is at 84%.

9. CONCLUSION

This paper discussed an extensible system for maintenance document OCR. Traditional off-the-shelf OCR solutions have inconsistent performance on tabular data. By segmenting the document into sub-images or cells and feeding that into an OCR engine results in consistent and accurate output. Furthermore this approach allows for granular level controls across all phases of the process from fine tuning the image segmentation phase to adding custom regular expressions based on document schema. Lastly this approach is scalable and adaptable to the various maintenance document schemas a product line may entail, ultimately reducing labor costs and increasing advanced analytics capabilities.

REFERENCES

Artifex Software. (2018). *Ghostscript.* Retrieved from https://www.ghostscript.com.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools.*

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*(6), 679-698.

Google Inc. (2017). *Tesseract.* Retrieved from https://github.com/tesseract-ocr.

Matas, J., Galambos, C., & Kittler, J. (2000). Robust Detection of Lines Using the Progressive Probabilistic Hough Transform. *Computer Vision and Image Understanding*, *78*(1), 119-137.

Patel, C., Patel, A., & Patel, D. (2012). Optical Character Recognition by Open source OCR Tool Tesseract: A Case Study. *International Journal of Computer Applications*, *55*(10), 50-56.

Smith, R. (2007). An Overview of the Tesseract OCR Engine. *Ninth International Conference on Document Analysis and Recognition*, *2*, 629-633.