Table Localization and Field Value Extraction in Piping and Instrumentation Diagram Images

Arka Sinha
Smart Data and Services
German Research Center for Artificial Intelligence (DFKI)
Kaiserslautern, Germany
Email: arka.sinha@dfki.de

Johannes Bayer
Smart Data and Services
German Research Center for Artificial Intelligence (DFKI)
Kaiserslautern, Germany
Email: Johannes.Bayer@dfki.de

Syed Saqib Bukhari
German Research Center for Artificial Intelligence (DFKI)
Kaiserslautern, Germany
Email: saqib.bukhari@dfki.de

***Abstract*—Piping and Instrumentation Diagrams (P&IDs) are graph-based engineering drawings utilised in process engineering. These documents also contain aditional information in tabular form. In this paper, the localisation and extraction of information of these tables are investigated. Documents used in this context are scanned raster version of P&IDs with tabular data inside a frame. The objective is to extract fields information from these tabular structures. This process is mainly divided into table localisation and then table field extraction from the segmented tables.**

The table localization task is achieved primarily with contour detection methods of computer vision. For the field-value extraction, a combination of rule-based keywords and navigation approach is used, utilising an Optical Character Recognition (OCR) for text extraction and regular expression for string comparison. This paper describes application of this extendable approach to the P&ID domain, where it achieved a promising result on a private dataset.

***Keywords*-Table Localization, Information Extraction, Piping and Instrumentation Diagrams**

I. Introduction

Piping and Instrumentation Diagrams (P&IDs) are an integral part of process engineering. These diagrams not only contain the graphical structure of a process engineering plant, but also complex tabular structures containing important information about the plant. In this paper, our focus is to segment those tables from P&IDs and extract information from those segmented tables. Character recognition technology has seen a lot of advancement through the recent years. There are already many well established methods for applying Optical Character Recognition (OCR) on a scanned document image. However, extracting text from an image like P&IDs where other non-textual information is present, is relatively complex as compared to text-only document images.

In this paper we are proposing a two step methodology for table understanding in P&ID documents. The first subtask is solely focused on localizing the tables i.e. to find out the coordinates of the tabular frames within a whole P&ID document image. Once tables are successfully detected, in the second subtask the desired fields can be extracted from the tables. Cropping out the tables beforehand reduces a huge amount of pre-processing load for an OCR engine and as a knock-on effect, it also reduces its probability of misreading characters since the amount of extraneous data is less. In the experimental section of this paper, it has been observed that the inaccuracy of OCR often undermined the result. Therefore, it was one of the primary targets to keep the scope of OCR as specific as possible.

This paper is further organised as follows. Section II briefly describe some previous work in the domain of table localization and information extraction. Section III describes the first step of our proposed methodology i.e. table localization in P&IDs. Section IV describes the tabular information extraction from P&IDs i.e. the second step of the proposed methodology. Both of these sections also contain their respective performance evaluation, results and their future work. A brief discussion about these results is described in Section V. Finally, Section VI concludes the paper.

II. Related Work

Tengli et al.,2004 [2], worked on HTML tables in webpages. They looked for ⟨`table`⟩ tags in the pages and parsed its contents. They analyse various tags (⟨`tr`⟩, ⟨`td`⟩, ⟨`th`⟩) to capture the table structure and their results were promising. Embley et al., 2016 [7], also worked on tables from the web. They segment and store tables for query processing and can deal with tables in wide variety of formats. However, since our work in this paper mainly deals with electronic documents with no backend HTML or XML, their method could not be adopted. Pinto et al.,2003 [3] applied Conditional Random Fields (CRFs) for table extraction. They label different lines in the images as per their relation with the tables. Then they trained their model to find out where are the table boundaries or header cell or row-column divisions. However, in P&IDs images the tables have frames and have contours for the cells. Hence, we propose to develop an algorithm without involving data labelling and training a model. The algorithm in this work is more inspired from the work of Riad et al.,2017 [1] where they also deal with scanned/digital images. They use connected component analysis for structural analysis of the image and use OCR for textual information extraction.

III. Table Localization Method in P&IDs

A. Problem Statement

Every document in this work has table which contains key information such as project number, project description, index etc. For the first part of the work, the objective is to detect the location of the table in the whole P&ID image. Figure 1 shows an example of a P&ID diagram. Similar images have been used as dataset for this paper.

One key feature in our dataset is that there is always one table in each image and it is always at the bottom half of the image. Since the scope of this work is primarily targeted for the current dataset, this observation is utilized to narrow down our search for table only in the bottom half of the image as shown in Figure 2a. Please note that the Figure 2a has been edited to anonymize the data and it is not the actual image used in this work.

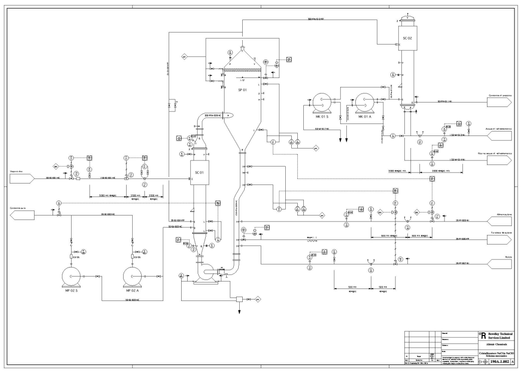


Figure 1: A sample P&ID Image [12]

However, we want our code to be applicable to as many general cases as possible. Hence, we edited some existing images to place the tables at random locations to test if the algorithm works when tables are located in various places. We also gathered images with multiple tabular structures to prove that the algorithm can detect more than one tables in single image.

B. Employed Technologies

The main programming language used for this work is `Python` (version 3.6) [11]. With `OpenCV` (version 3.4.1) [9] library of Python, programmers have access to a wide range of pre-built functions for computer vision techniques. `Tesseract` (version 3.05.02) [10] OCR engine has been used to carry out the part of field value extraction. Python also supports packages for string matching (using regular expressions RegEx). The target of this work is to use these powerful tools in the right way to get the desired results.

C. Methodology

The non-table rectangular graphical structures usually contain less textual information and less internal cellular structures which is found in table because of the intersection of row-column borders. Hence, to localize the tables, the algorithm starts by looking for the rectangles in the image. We use `findContours` method from OpenCV package as the primary means to find all the edges or contours in the image. This function can find the lines joining continuous points of same color [8]. Among several options of contour retrieval, we settled for retrieval mode as `RETR_LIST` and approximation mode as `CHAIN_APPROX_NONE` to ensure that we get each and every contour without any compression and also we want to avoid any internal hierarchy between the contours. While contours are being retrieved, they are passed to `approxPolyDP` method of OpenCV to filter in only the rectangular shaped contours.

D. Main Issue

Tables are made of multiple adjacent smaller rectangles created by intersecting rows and columns. One of the major issues faced during this phase was that the `findContours` function was detecting the contours of the internal rectangles but not the contour of the main table as a whole. Even using `RETR_EXTERNAL` as retrieval mode did not give the desired result. Part of the problem was that in most cases the table shared two common boundaries with the image margin. Therefore, the first target was to remove the margin. Our program takes width of the margin as parameter. It draws an unfilled rectangle with the same dimension as the image with white border. The width of the border will be same as the margin width parameter and the resulting image will be the whole image without the margin lines. In this paper, after analysing the available dataset, we settled on 0.015 (1.5 % of the total image width) as margin width and it gives us the desired result.

E. Table Blackout

Once the margin is removed, next target is to find a way for the function to recognize the outer boundary of the table as one single contour. Since the problem was due to the function recognizing only the inner rectangles, it was necessary to somehow remove them from the image. For this a unique approach has been followed. We create a new white image object with the same dimension as the input image. Whenever the function detects any rectangle in the original input, it draws a filled black rectangle in that position with the same dimension (of the detected rectangle) in the white image. At the end of the full iteration, it results in to the image shown in Figure 2b, where only objects in the image are patches of black blobs in place of detected rectangular shapes. Now that the program has made the table or any other rectangular structures into patch of black rectangles, this resultant image is again passed through the process of contour detection. This time since no other non-rectangular graphical structures is present, the process becomes much easier and accurate for the `findContours` function.

F. *Table Selection*

Now that the coordinates of the rectangles in the image are known, another mechanism is required to identify the actual tables among them. In our dataset it was apparent that the non-table rectangular graphical structures usually contain less textual information and less internal cellular structures which is found in tables. Hence to differentiate between tables and other rectangular symbols, the program at first counts the number of internal cells within the rectangles. If the count of cells is above certain number, then it is a possible candidate for table. The program also accepts rectangles as tables if it has more than certain amount of textual information. For performance improvement, while detecting the amount of text, we avoided using `Tesseract` OCR engine since we don't need the actual information of the text in this section. Instead, the program measures the amount of contours it can detect within the rectangles. Characters and letters have their own contours. Hence if we add up the areas of detected contours in the tables then it will be more than the area of the actual rectangle. Using these two validation criteria, our program differentiates tables from all other rectangular structures.

G. *Results*

The function finally returns the coordinates of the top-left corner point and the width and height of the table. This algorithm was executed on 106 images of P&IDs from a private use case where each file has at least one tabular structure. Since the tables were always situated at the bottom half, for performance improvements, we applied the algorithm on bottom half of the image only. As shown in Table I, the code was able to segment 109 tables from the images successfully. For 4 images, the tables were not fully connected to the margin and also had no other boundary around the table. For those images, the algorithm could not detect the whole table and identified the table till its last found column boundary. There were 2 images where the proposed algorithm did not work as it failed to identify any table from them and they were not used in subsequent steps. Lastly, we got 10 rectangles which were not tables but were wrongly segmented from the images.

Table I: Table detection result statistics

Correctly Identified Tables	Partially Identified Tables	Wrongly Identified Tables	Not identified Tables
109	4	10	2
87.2%	3.2%	8.0%	1.6%

For testing the generality of our code, we tested our program with an image with 6 tables and it detected all of them. We also tested our code with some edited images from the dataset where the location of the table is random and it was still able to segment the correct tables.

Since `approxPolyDP` method is used for rectangle detection, which is an approximation method, the program can detect tables which are not a perfect rectangle as well.

H. *Future Work*

Knowledge of the dataset has been used in some places to tune the algorithm for better performance. However, this algorithm for detecting table can be easily adapted for other dataset with some changes. Some of the most likely changes that may be required are as follows:

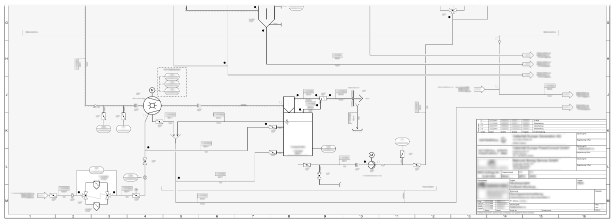
- This algorithm takes margin width as a parameter. For new set of images, a new value for the margin width needs to be assessed and passed on.
- Since all of the tables in the dataset are at the bottom half of the images, the algorithm is applied in that region only for better performance. Half of the image means half the number of pixels to analyse. There is no other reason behind this decision and we have tested that our algorithm can seamlessly work on full images as well even if the location of the table is fully random.
- The algorithm is designed to select a rectangle as table candidate based on how many cells it has and how much other contours (assuming most of them are texts) it has inside. The threshold of these parameters has been set heuristically for our dataset. However, these threshold values may need modifications for other datasets.

As it can be seen from above, all of these changes may require some alteration in scripts, but the core algorithm, will remain the same.

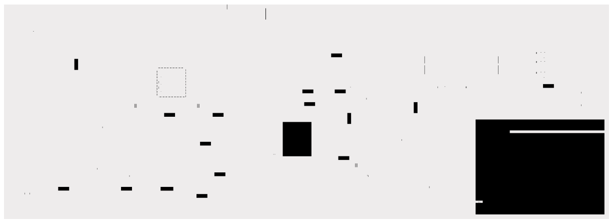
IV. Tabular Information Extraction Method from P&IDs

A. *Problem Statement*

After detecting the tables in the image, the next task is to extract specific information from them. The challenge during this phase is that the location of a field value with respect to its key or header is not fixed. For example, some of the field-values (e.g. the name of the processing plant) can be extracted without evaluating the content of their neighbour cells. In contrary, for extracting the correct version of the P&ID from the table, the content of the last non-empty cell

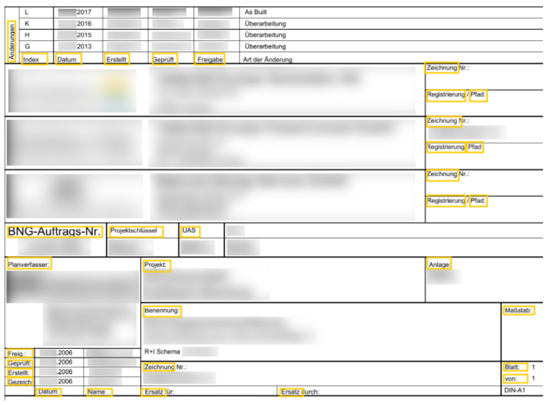


(a) Input Image (bottom half)



(b) Rectangle detection result

Figure 2: Illustration of table segmentation (Data anonymized). Background has been darkened for illustration purpose.



Änderungen						
	L	2017				As Built
	K	2016				Überarbeitung
	H	2015				Überarbeitung
	G	2013				Überarbeitung
	Index	Datum	Erstellt	Geprüft	Freigabe	Art der Änderung

Zeichnung Nr.:
Registrierung / Pfad:
Zeichnung Nr.:
Registrierung / Pfad:
Zeichnung Nr.:
Registrierung / Pfad:

BNG-Auftrags-Nr.	Projektschlüssel	UAS	

Planverfasser:
Projekt:
Anlage:
Benennung:
R+I Schema
Maßstab:

	Datum	Name
Freig.:	2006	
Geprüft:	2006	
Erstellt:	2006	
Gezeich:	2006	

Zeichnung Nr.:
Blatt: 1
von: 1
Ersatz für:
Ersatz durch:
DIN-A1

(a) Table candidate 1

Vorlagebehälter

(b) Table candidate 2

Figure 3: The two segmented rectangles from the example above. Based on the high number of cells and textual data, candidate 1 is selected as table. The keywords later used for field extraction is highlighted (yellow boxes). Sensitive data has been anonymized.

has to be used as value. Therefore, even if we could write dedicated functions for extracting each field separately, it would have made future extensions very difficult because if any new fields are to be added whose locations might be different than the previously extracted fields, we will have to write a separate function.

B. Methodology

To avoid such scenarios, we opted to develop our algorithm in more generic way. We decided to separate the information about location of the fields from the methodology to traverse to a particular cell as per requirement. Therefore, the program for this part has two components:

- Configuration files containing the information about the fields to be extracted and where to look for the value.
- Python scripts that read these configurations and act accordingly

C. Experimental Setup

Our dataset contains three types of tables. Each type of tables has exactly similar layout but very different from other types. Therefore, instead of entering configuration for each file separately, we instead entered configuration for each type of tables. Hence, we managed to encode configuration for 104 files into 3 entries (3 separate configuration files). The configuration files contained fields' name which are to be extracted and the location of the cell where our algorithm is required to search for the fields' value.

The Python scripts first have to decide which type out of the three it is working with. For this task we take a naive approach of comparing the number cells of each type. Since in our dataset, the layout of each type is completely different, this method of categorisation works accurately. After the table's type is detected, the script looks into its corresponding configuration file to get the information about which text or word to search for as field's key and once the key have been found, it traverses in the direction mentioned by the configuration for the value.

To make the traversal efficient and manageable, we decided to transform the table into a network graph. Each node will represent a cell of the table with the text inside the cell and the width-height, coordinates of the cells are represented by the node properties. The adjacency of cells in the image will be replicated in terms of edge connectivity in its graph form. The label of the edges signifies the relative directions of the cells/nodes. This method of converting table image into a network graph makes it easier to hop either one or multiple nodes in particular direction. The flow diagram in Figure 4 shows how our algorithm works. Most of the fields' value can be retrieved with these methods. However, we had to address two special cases that we encountered and wrote dedicated methods to solve.

The first case is where the fields' key and the fields' value is in the same cell. Which means that the text we extract to check for the fields' key, also contains the value itself. Hence, we had to develop an algorithm to remove the key from the extracted text and keep only the value.

The other case is for fields like "Index" or "Rev", we only require the latest value. Our aim is to traverse through all values and take the last one. Therefore, while traversing through the table graph, our program checks if there is any neighbour node in that direction or if the neighbour node has no text. Either cases signify end of search and returns the last found value.

D. Employed Technologies

JavaScript Object Notation (`JSON`) files have been used to store the field extraction rules. JSON is natively supported by Python and provides sufficient methods to easily access the required data from a JSON configuration files.

In Python scripts, we again use `findContours` method from OpenCV package to detect all the cells within a table and get their coordinates, height and width. While converting the table into a network graph, we used the `NetworkX` (version 2.3) library for Python. For extracting the text within a cell, we opted for `Tesseract` OCR engine and then for matching those text with configuration information, we use regular expressions.

E. Results

A total of 104 cropped table images from the first subtask has been selected for field extraction. We decided to extract 5 different fields for each type of tables and judge the performance based on how many out of those 5 could our program recognize. Please note that the accuracy of the extracted text (as field value) was not considered as it is heavily dependent on the internal algorithm of Tesseract. For some cases Tesseract reads "Z" as "2" or "I" as "1". The focus was mainly on whether the program can detect the field key in the table and retrieve the corresponding value from the location mentioned in the configuration file. Based on such criteria we observed that for 29 tables, our algorithm was able to successfully extract all 5 fields. For the rest of the tables we were able to detect at least 3 or more fields. Figure 5 shows the result of our algorithm at its current state.

Some manual intervention was required to improve our result in some places.

- Since Tesseract was often confusing between letter "I" and digit "1", for the "Index" field we explicitly mentioned to search for "Index" or "1ndex".
- For the 4 cases where tables were partially detected, we had to change the field search key to account for the missing characters.

Please note that these manual changes were only restricted to JSON configuration files and the Python scripts were not altered in any way to address such special cases.

F. Future Work

Similar to the first part, we have used our knowledge of data to foresee some special scenarios and designed our configuration files accordingly. However, by keeping rules for retrieval separate from the actual methodology of traversal, we have tried to ensure that any new addition of fields should only require extra entry in the configuration file. In spite of this, there are multiple areas where our algorithm can be improved:

- Since our three categories of tables are vastly different to one another, comparing the number of cells to categorise was sufficient. But if in future a new type of tables comes into input list, then we need to find a better method (e.g. applying SVM or PCA or other machine learning algorithm) to assign a more unique signature for the tables to differentiate their types.
- If an entirely new type of table is given as input, we need to write a separate JSON file and also add lines in script to generate its own contour signature.
- Our method is highly dependent on the accuracy of Tesseract output and hence in future we can implement any better alternatives.
- We have not addressed the case of having more than one key-value pair within a single cell.

V. Discussion

The proposed algorithm for table localization works fairly accurately as we were able to crop out the correct tables from the images only apart from two images. There were some extra rectangles labelled wrongly as tables which we will work on to filter out better in future. In information extraction part, the results have been mixed. As per our analysis, the presence of graphic elements including some containing text like logos, may have led to OCR errors and require more filtering to ignore them. In some cases, the letters were not aligned properly and was touching cell boundary which is also difficult for OCR to read. One must also have to factor in the probabilities of Tesseract misreading some characters because of its internal approximation algorithm. They all contributed to some inconsistencies in our result.

VI. Conclusion

This paper primarily uses computer vision techniques, text processing and Optical Character Recognition technology. We had to use our knowledge of dataset in some places to get more accurate output. Logically this algorithm should still adapt well to other dataset with very little changes. In future

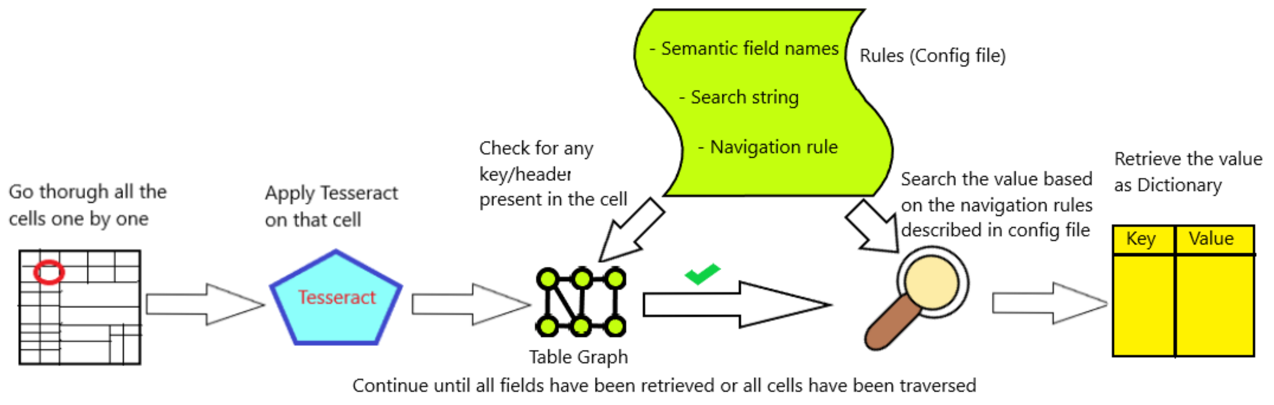
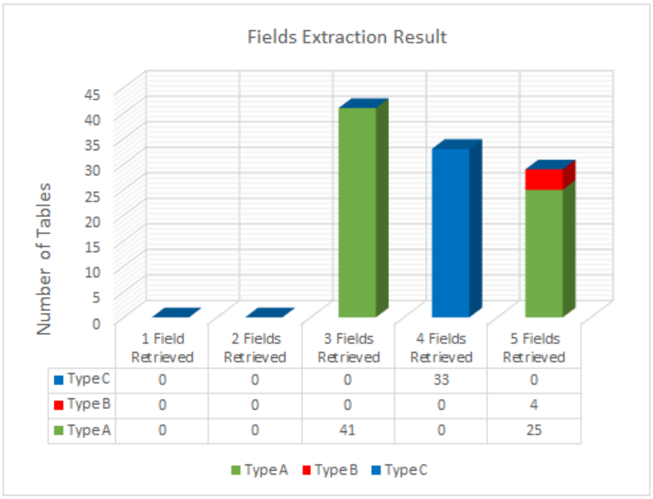

Go thorugh all the cells one by one
Apply Tesseract on that cell
Tesseract
Check for any key/header present in the cell
Table Graph
- Semantic field names
- Search string
- Navigation rule
Rules (Config file)
Search the value based on the navigation rules described in config file
Retrieve the value as Dictionary
Key
Value
Continue until all fields have been retrieved or all cells have been traversed

Figure 4: Fields value extraction workflow.



	1 Field Retrieved	2 Fields Retrieved	3 Fields Retrieved	4 Fields Retrieved	5 Fields Retrieved
🟦 Type C	0	0	0	33	0
🟥 Type B	0	0	0	0	4
🟩 Type A	0	0	41	0	25

Figure 5: Fields value extraction result

we would like to develop a complete generic algorithm with minimal human involvement across various dataset. Currently our program can only detect tables inside frames, but we want to extend our algorithm to detect tables without borders as well. We think machine learning algorithm can help us achieve that target. We can use state of the art semantic segmentation networks (e.g. R-CNN [4], SegNet [5], ResNet [6] etc.) and adapt them for our use cases. If we can successfully label the pixels as background pixels and table pixels separately, then we can segment the table from P&IDs irrespective of their borders. Although the data set for this type of work is limited, we can use data augmentation to increase our training dataset. A successful combination of such methods can eliminate the need of any manual input and potentially perform more accurately.

References

[1] Riad, Amir, et al. “Classification and Information Extraction for Complex and Nested Tabular Structures in Images.” 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR). Vol. 1. IEEE, 2017.

[2] Tengli, Ashwin, Yiming Yang, and Nian Li Ma. “Learning table extraction from examples.” Proceedings of the 20th international conference on Computational Linguistics. Association for Computational Linguistics, 2004.

[3] Pinto, David, et al. “Table extraction using conditional random fields.” Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval. ACM, 2003.

[4] Girshick, Ross, et al. “Rich feature hierarchies for accurate object detection and semantic segmentation.” Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.

[5] Badrinarayanan, Vijay, Alex Kendall, and Roberto Cipolla. “Segnet: A deep convolutional encoder-decoder architecture for image segmentation.” arXiv preprint arXiv:1511.00561 (2015).

[6] He, Kaiming, et al. “Deep residual learning for image recognition.” Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[7] Embley, David W., et al. ”Converting heterogeneous statistical tables on the web to searchable databases.” International Journal on Document Analysis and Recognition (IJDAR) 19.2 (2016): 119-138.

[8] Contours : Getting Started, Retrieved March 12, 2019, from https://docs.opencv.org/3.3.1/d4/d73/tutorial_py_contours_begin.html

[9] OpenCV library, Retrieved March 12, 2019, from https://opencv.org/

[10] tesseract-ocr, Retrieved March 12, 2019, from https://github.com/tesseract-ocr/

[11] Welcome to Python.org, Retrieved March 12, 2019, from https://www.python.org/

[12] Piping and instrumentation diagram, Retrieved March 11, 2019, from https://en.wikipedia.org/wiki/Piping_and_instrumentation_diagram#/media/File:P%26ID.JPG