AN INTELLIGENT AND AUTOMATED PDF FORMAT CHECKER

A Project

Presented to the

Faculty of

California State Polytechnic University, Pomona

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

In

Computer Science

By

Tsaihsuan Yang

2017

SIGNATURE PAGE

PROJECT: AN INTELLIGENT AND AUTOMATED PDF FORMAT CHECKER

AUTHOR: Tsaihsuan Yang

DATE SUBMITTED: Spring 2017

Computer Science Department

Dr. Yu Sun _____
Project Committee Chair
Professor of Computer Science

Dr. Mohammad Husain _____
Project Committee Member
Associate Professor of Computer Science

ABSTRACT

PDF (Portable Document Format) document, unlike Microsoft Word file which can check format by parsing its XML metadata, is hard to locate the format settings, as well as text processing- It is not an easy task to check the formatting compliance of a PDF document due to the challenge of retrieving the settings metadata. To check a PDF document format manually is time-consuming and error-prone. In this paper, we have designed and developed an intelligent system to automate the format checking and the compliance verification. This system can help reviewer to examine format in a short time with good format error detection.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

PDF (Portable Document Format) [1], is a file format that is widely used now. The PDF format is a cross-platform format, people can view a PDF document in various devices without losing format. As a result, PDF format is very suitable for formal paper or document submission. When you submit your final paper to conferences, to your school, they usually have format requirements such as page size, page margins, font styles, and so on.

To ensure documents meet those format requirements, reviewers would have to check manually if there is no software to help them do this task. Checking the format of a PDF document manually is time consuming and error-prone. For example, in a PDF document, it is not feasible to determine if the margins of a page meet the requirements by looking at the document manually. Additionally, it is not trivial to manually distinguish between font families such as “Times New Roman” and “Georgia”. Table 1 shows two sentences with similar font styles, it may be hard to identify their font family.

Times New Roman	The quick brown fox jumps over the lazy dog
Georgia	The quick brown fox jumps over the lazy dog

Table 1 Sentences with similar font styles

To check format settings automatically, first we need to obtain the metadata of format settings of a PDF document. However, unlike documents such as Microsoft Word, to retrieve format metadata of a PDF document is not straightforward. Since PDF format was initially designed for viewing and printing, people cannot easily edit a PDF document, not mention to retrieve format settings or data from a PDF file.

To solve this problem, we use a tool called PDFBox to help us retrieve the format settings information. Furthermore, we can use the information generated from PDFBox to design a system that can check PDF format.

In our paper, we propose a system that is able to automatically check the format of a PDF document (e.g., graduate thesis, conference publications) using template matching. The typical format requirements include: paper size, page margins, font style, line spacing, pagination, sequence of parts. Besides, if this document has Table of Content, we should check its correctness. Our experiments show that the system can indicate the invalid format to those typical requirements in a short time (it takes less than 15 seconds to check a 40-page document). Instead of reviewing the PDF document directly by manual, users now can simply read the checking report generated by our program and find the format problems easily.

The remainder of the paper is organized as follows. In Section 2, we provide motivating example and challenges when doing format checking only by manual. In Section 3, we introduce our solution- PDF format checker with its details. We cover our experiments in Section 4, and introduce a related work in Section 5, and finally, we conclude our work in Section 6.

CHAPTER 2

MOTIVATING EXAMPLE AND CHALLENGES

In this section, we will introduce some challenges that checking manually can have. Some examples of the detailed formatting requirements are as follows:

Item	Example Requirement
Paper size	Letter size
Page margins	Left: 1.5”, other: 1”
Font style	Between 10 to 12 points, Times New Roman
Spacing	Double space
Pagination	Pages before main body of the document should use Roman numeral (ii, iii, iv…). Start from main body, it should use Arabic numeral and starts from 1.
Sequence of Parts	Title page, Signature page, Acknowledgments, Abstract, Table of Contents, List of Tables, List of Figures, Text, References, Appendices
Table of Contents	Each item in table of contents must be showed in the target page correctly.

Table 2 Example requirements of format

Only page size can be checked correctly, since you can have the exact value of page size by checking PDF document properties from a PDF viewer. However, other requirements are very difficult to see if they are valid or not, because PDF viewer cannot provide you enough information to check.

The most difficult task within these requirements is checking Table of Contents. To do this task manually, first, you need to take a note of all the items in Table of Contents and its target page number (the page number following the item). Then, you will need to convert this page number to a “real” page number that you use this value the jump to the real target page. Finally, you go to the target page and see if the item is existed or not.

Other requirements are not easy to check either. For example, the page margins, font size, line spacing, you may use the tool from a PDF viewer as well, but it is not straightforward. Font family, as we mentioned in Section 1, similar font family is hard to distinguish.

The challenges for checking format, as we discussed in Section 1, it is difficult to gather formatting information of a PDF document, hence, we need an additional library - PDFBox to assist this functionality.

To check format manually, in average, it would take 7 to 10 minutes. It is a tedious work, and sometimes you cannot find a very small difference by your eyes. Hence, our goal is to solve the problems of manually check format, and provide a system to handle this problem in an automated way.

CHAPTER 3

SOLUTION

To solve the difficulties and make checking process easier and correctly, we present a solution – PDF format checker, to make sure the format of your PDF document meets all the requirements. Our program first can parse all the glyphs' information, such as font style, the left and top position of a glyph. With these information, we combine them into more useful and straightforwardly information such as margins value, line spacing, etc. As we discussed in Section 1, to get formatting information of a PDF document is not easy, hence, we need to rely upon an additional library to help us parse the glyphs' information. The library we are using is Apache PDFBox, which can parse the text information glyph by glyph – glyph is the smallest unit (that is, a character) in a PDF document.

3.1 Program Structure

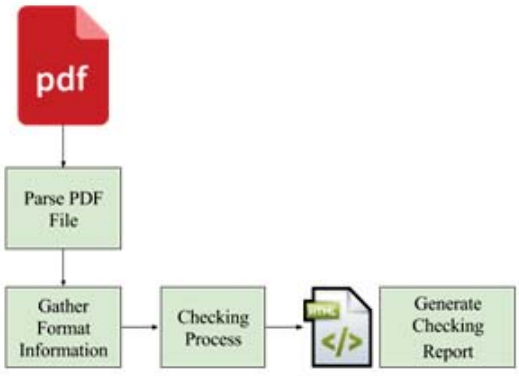

Figure 1 Program structure of PDF format checker

The program uses PDFBox to parse a PDF document, in the stage "Gather Format Information", it will collect information for a page includes margins, page width and height, page number, base font style, etc. The checking process includes the format check

for each page, as well as sequence of parts and table of contents correctness. Finally, the report will be presented in HTML format.

3.2 Data Collection

The first step is to collect enough information for each page. Which includes: top and left position, font family, font size and direction. We also need page size, that is the mediabox dimension values, and retrieve the page number.

3.2.1 Find Page Size

PDFBox can provide this information for use, simply check the dimension values of page's mediabox. The unit of value is pixels. To convert it to inches, we can use the following conversion:

$$inches = pixels / 72$$

3.2.2 Define Margins

We use text's position to identify margins of a page. To find top and left margins, we find the line with smallest top and left position values. The position values are the margins values.

To find bottom margin, first we need to locate the line with the second large top position. The reason why the bottom margin does not take the largest top position is that, the largest top position is usually the location where page number is. However, when we talk about bottom margin, we do not count page number. The second large top position of text, which is usually the location where the last line is. After we obtain the second large top position, there is an additional calculation we need:

$$bottomMargin = pageHeight - secondLargeTopPosition$$

Similarly, defining right margin is to find the text position which has the largest left position, and apply a similar formula to calculate right margin:

$$rightMargin = pageWidth - largestLeftPosition$$

Since the unit of values are pixels, we need to convert it to inches. We use same approach as we talked in 3.2.1 Find Page Size.

3.2.3 Define Base Font Style

Base font style is the font style includes font family and font size that are mostly used in a page. There might be different font styles in a page, to define base font style, we first find how many different font styles used in this page, afterwards, we calculate the frequencies of each font styles in a page. The unit we are looking for is a glyph. For example, the font family "Times New Roman" appears 243 times, and font family "Calibri" appears 5 times, we pick Times New Roman as the base font family in this page. To define base font size, we use same approach.

3.2.4 Define Spacing

To define spacing used in a page, we need to find the top positions of each lines. Then, to calculate spacing between lines[i] and lines[i-1] is simply

$$lines[i].topPos - lines[i-1].topPos$$

Similar to what we do for finding base font style, we can calculate the spacing used most often as the base spacing of current page.

3.2.5 Find Page Number

As we mentioned in section "3.2.2 Define Margins", the last line's top position cannot be used as bottom margin, since the last line of a page always is page number. Hence, to find page number, we can simply pick the text content of last line.

However, sometimes author would miss page number, in this situation, last line would be text of paragraph and cannot be used. To solve this problem, we calculate the word count for last line. No matter the page number is a Roman numeral or an Arabic numeral, there should be only one word. Hence, if the word count of last line is not equal to 1, we recognize this last line is not a page number and will leave the page number blank. When the word count of last line is equal to 1, we need to do an additional check: to make sure it is the page number we want, not a random character. To do this check, we can try if this word is convertible to an integer or roman numeral. If not, then this value is not a valid page number we can use.

Finally, we need to know the difference of PDF page number and real page number. Table 3 shows the definition of PDF page number and real page number.

PDF page number	Real page number
males between the ages of 19-50 (Institute of Medicine, 2005). According to the National Health and Nutrition Examination Survey (NHANES) statistics and the trends that documented the dietary fiber intake from participants from 1999-2008, the average fiber 3	

Table 3 PDF page number and real page number

Real page number can be fetched by PDFBox simply.

Understanding the PDF page number and real page number helps us to do the pagination check, we will discuss the checking process in the following section.

3.2.6 Find Sequence of Parts

Before we find main body start page, first we need to make sure the sequence of parts meets the requirement. To do so, we need to find the page numbers where the parts locate. For example, the part “ABSTRACT” locates on page 3, the part “REFERENCE” locates on page 43, etc. After we fetch all the parts with page numbers, we can check if

the document meets the sequence of parts requirement.

The approach to do this, we look for the first line of each page. If the first line's text equals to the names of required part, then this line is the heading we want to use:

```
String [] seqOfPartsNames = ["Signature Page", "Abstract", "Acknowledgement",
"Table of Contents", "List of Tables", "List of Figures", "References", "Work Cited",
"Bibliography", "Appendix"]
// find headings and whose page number:
for (Page p : allPages)
    for(String s : seqOfPartsNames)
        if(p.firstLine.equals(s))
            heading.pgNum = p.pgNum
            heading.text = p.firstLine
```

Finally, we will have a list containing all the headings and the real page number. Afterwards, we can check the sequence of parts is valid or not.

headings[0]	["Signature page", 2]
headings[1]	["Abstract", 3]
headings[2]	["Table of Contents", 4]
…	…
headings[n]	["Appendix", 70]

Table 4 Information of headings

Since some of the part can be optional, to check sequence of the parts, we can use the concept "if part B is reachable to part A?". The following is an example relationship

between parts:

Part	Can reach to
Signature page	Abstract or Acknowledgment*
Acknowledgement*	Abstract
Abstract	Table of Contents
Table of Contents	List of Tables* or List of Figures*or References
List of Tables*	List of Figures* or References
List of Figures*	References
References	Appendix*
Appendix*	Appendix*

Table 5 Relationship between parts. Items with * means they are optional

We can convert this table to a graph. If a node N has out edges to M1, M2, M3…Mi which means node Mi is reachable from N.

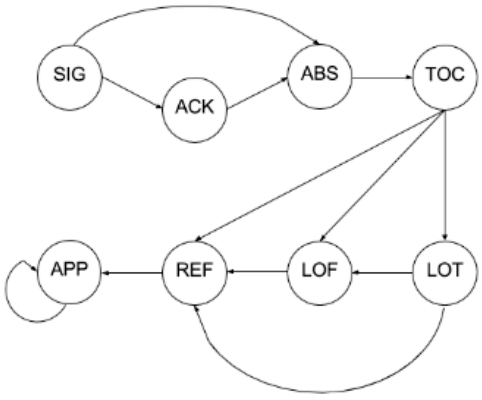

Figure 2 Relationship between parts in graph

If the headings[n].outEdges contains headings[n+1], which means headings[n+1] is reachable from headings[n], and this is valid. Otherwise, if headings[n].outEdges does not contain headings[n+1], that is, the sequence is wrong between these two.

3.2.7 Find Main Body Start Page

To define main body's start page is important for checking pagination. We will explain this importance in section "3.3.3 Pagination".

Main body start page is the page which contains like "Chapter 1", "Introduction", etc. No matter what the heading of the main body is, this page must be appeared right after Table of Contents (TOC). Although author may have List of Tables and/or List of Figures after TOC, the formats of them are same, hence, the TOC we talk about here also includes "List of Tables" and "List of Figures".

Remember that we have already gathered the headings information (Table 4). Now, we know where the start page of TOC is. To find where the main body page starts, we can follow this approach:

```
int searchRange = min (25, totalPageCounts/5)
for (int i=TOCIndex; i<TOCIndex+searchRange; ++i)
    if(allPages.get(i).isContentStart())
        contentStartPgNum = i
        break
```

The equation for searchRange is proposed by [5]. They used this to calculate the Kth page where the TOC page might end. In our program, we use this number as same propose, however, [5] choose a minimum from (20, totalPageCounts/5), here, we use 25 instead because when we use our test data with more than 200 pages, using 20 as our searchRange may not locate the main body successfully as there would be lots of tables or figures in TOC list.

The function isContentStart() is to determine whether if the page is the first page of main body.

A page is a first page of main body if:

- This page locates within potential TOC page range.
- This page is the first page which has characteristics of main body.

Hence, the problem becomes: what is the characteristics of main body? After evaluating the documents, we found that main body has the feature that the text ratio of right part of a page will be much higher than the number ratio. Taking right part of a page is because the right part of a TOC page usually contains numbers. Hence, when text ratio is larger than number ratio, we can infer that this page is main body.

When complete this step, we should be able to locate the real page number of first page of main body.

3.3 Check Page Format

Section 3.2 introduced how to retrieve the prerequisites for us to do the checking process. Now, we can use this information to deal with checking process.

Here are the sample format requirements for our experiments:

- Page size (Should be letter size)
- Margins (top/bottom/left/right: 1/1/1.5/1 inches)
- Line spacing (Double spacing)
- Pagination
- Font family and size (valid font family includes: Times New Roman, Arial, Calibri, size between 10 to 12 points)

3.3.1 Check Page Size, Margins and Font Styles

To check page size, margins and font styles are easy, since we have retrieve these data already. All we need to do is to compare these data with standard requirements.

To check if author uses different font styles in a page, we compare each glyph with the base font style and store glyph's information when it does not match base font style. Afterwards, we can use the different font styles glyphs' information to create an image to let user see the locations of different font styles used.

3.3.2 Line Spacing

As we talked in section 3.2.4 Define Spacing, we use each line's top position to define spacing used. However, different font size yields different standard to check line spacing. To understand different standards, we create different PDF documents with different font family and font size, and examine these test files' line's top positions, and we have the summary for defining standard for double spacing as follows:

Font size (points)	Standard for double spacing
10	21-26 pixels
11	23-28 pixels
12	26-31 pixels

Table 6 Line spacing standards

When line spacing requirement is different, we can use same approach to find the standards.

3.3.3 Pagination

In section "3.2.5 Find Page Number", we discussed how to fetch PDF page number and real page number on each page.

Pagination check is to make sure page number sequence is correct. To do this task, we need: page number pair of first page of main body and page number pair of every page.

The page number pair (PDF page number, real page number) of first page of main body, the PDF page number will be 1, while the real page number can vary. For example, (1, 13) is the page number pair of first page of main body. After we know the difference between these two numbers, we can apply this difference to all pairs of page number to make sure if the pagination is correct or not. For example, with this difference, when we find (2,15), then the pagination check result of this page will return a false.

A formula convert between PDF page number and real page number is:

$$realPgNum = PDFPgNum - 1 + mainBodystartPgNum$$

3.4 Check Correctness of Table of Contents

Table of Contents includes “List of Tables” and “List of Figures”, since they share same format. The goal of this section is to retrieve items in TOC and check if the item is existed in the target page.

In section “3.2.6 Find Sequence of Parts”, we already located the real page number of the page that TOC starts. The pages between TOC start page and first page of main body should be TOC format. To make sure these pages are TOC format, we can use the approach proposed by [4]: TOC page usually has a sequence of numbers in the right part of page, and these numbers are increasing order. Within these TOC pages, we can start to retrieve table items information. Table items information should have: item name and its target page.

Signature Page ii

Table item name	Target page number
Signature Page	ii

Figure 3 Convert text on PDF into useful information

We can separate a line into three parts: table item name, dots (or other symbols), target page number, and store item name and target page number for later use.

However, sometimes the table item name is long, so that we cannot find a target page number within same line. Figure 4 shows this problem.

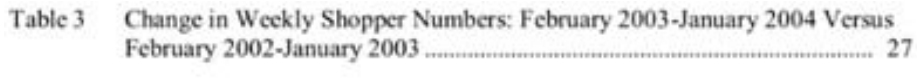
Table 3 Change in Weekly Shopper Numbers: February 2003-January 2004 Versus February 2002-January 2003 27

Figure 4 Problems of TOC item retrieving

Our solution to this problem is to assign a "NEXT" when target page number is not found. After first round of retrieving table information of TOC page, we should have things as follows:

Table item name	Target page number
Table 1: table1 table1 table1 table1	11
Table 2: table2 table2 table2 table2	NEXT
table2 table2 table2 table2	NEXT
table2 table2 table2 table2	15
Table 3: table3 table3 table3 table3	NEXT
table3 table3 table3 table3	22

Table 7 After first round of TOC items retrieving.

It is clearly that the table 1 has target page number of 11, while table 2 and 3, since the table item name is wrapped to next line, we cannot find target page number, we assign a "NEXT" instead.

Then, we can assign a correct target page number of items with "NEXT" by grouping them together with the following function.

```
GroupFunction(items):
    if(items[i].targetPgNum == “NEXT”):
            if(isNumber(items[i+1].targetPgNum))
                items[i].targetPgNum  = items[i+1].targetPgNum
    GroupFunction(items)
     return items
```

Table item name	Target page number
Table 1: table1 table1 table1 table1	11
Table 2: table2 table2 table2 table2	15
table2 table2 table2 table2	15
table2 table2 table2 table2	15
Table 3: table3 table3 table3 table3	22
table3 table3 table3 table3	22

Table 8 Completed TOC items information

Since the target page number is a PDF page number, we need to convert it to a real page number.

$$realPgNum = targetPgNum - 1 + mainBodystartPgNum$$

Then we get the Page content and check existence:

```
Page = allPages.get(realPgNum)
if (item is in Page): return true
```

3.5 Report of Checking Results

After checking process, the program will give user a report of checking results, which includes html files and images. Images contain the text using invalid font styles, since it is the most difficult part for human eyes.

CHAPTER 4

EVALUATION

The dataset we used for evaluation is the PDF documents published by the Cal Poly Pomona Library. Before these documents are published to website, it should be checked by manual, however, after we run this program to the PDF documents we download from website, there are still invalid format detected. Actually, those invalid formats are usually small and hard to see by eyes, the following image is one of the results by our program:

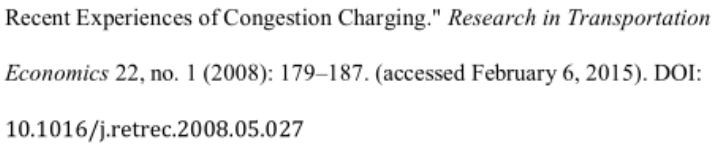

Figure 5 Origin content

Notice the last line of this image is using different font family.

Figure 6 Checking result image showing font family is wrong

We performed 70 PDF documents and our program can handle 64 of them: the program can read the file and do the check process and give us a good result. The other 6 files cannot be processed is because they were created by Latex. Document created by Latex usually has font family that cannot be parsed by PDFBox, since the information cannot be retrieved well, the checking process cannot perform good results.

The most important part is to define the first page of main body of document, we cannot check pagination and TOC correctness without this value. Besides, as we discussed before, the most tedious work is to check TOC correctness, hence, our evaluation will focus on (1) correctness of page number of start page of main body, (2) TOC items

retrieving correctness, and finally (3) performance evaluation. For evaluation (1) and (2), we compare the retrieved values with origin PDF document by manual. For (3), we record program running status then evaluate.

For (1), the correctness rate of locating page number of start page is 87.5 %, which is, we correctly defined 56 documents out of 64.

For evaluation (2), TOC items retrieving correctness rate is 99.14 %, when all TOC pages range can be fetched. That is, when the program can detect first page where main body starts, TOC items can be retrieved well. Situation when retrieving TOC items fail is because author's TOC format is not what we expect, or the target page number is too close to the item's text, hence we fail to retrieve the target page number.

If first page of main body detection is failed, there would be some TOC pages are lost. Depends on how many pages of TOC are lost, the TOC items retrieving rate can vary, the lowest rate is 36.9%, highest rate is 92.16 %. We can conclude that first page of main body is essential part for our program.

We did experiments on a laptop with 64-bit Windows 10, intel i5 CPU (2.30 Ghz), 8 GB RAM. The average number of pages of a document is 96.1 pages. The largest document has 543 pages, which takes 85 seconds to process. In average, to complete a checking process takes 14.7 seconds.

CHAPTER 5

RELATED WORK

Banal [6] is an online format checker designed by Geoffrey M. Voelker from University of California, San Diego. In this program, it can do a simple format processing. The results of Banal include the margins, base font size, columns in current page, while it does not provide checking comparison. The approaches of Banal is using template matching as well.

Compared with Banal, our work has abilities to do more checking processes. For font style checking, our work can locate those using invalid styles and generate an image to mark the location. Besides, when a document has table of contents, we can check its correctness. Moreover, we proposed an algorithm to check if sequence of parts is in order. As long as we can retrieve headings and its location, we can do this checking process.

CHAPTER 6

CONCLUSION AND FUTURE WORK

After the experiment we did for the documents, our program does provide decent checking results. While the checking process can be done by manually, it is really time consuming. Our program not only saves time on checking process, but also improve the correctness of checking results a lot comparing with checking by eyes. Finally, our program provides user a well-formed checking results in html files.

However, there are still some problems need to be refine, for example, the program cannot deal with documents created by Latex currently as PDFBox cannot parse font family used by Latex, and which results in errors. To solve Latex documents problem, we can directly parse Latex source files instead, which is an easier approach compared with parsing a PDF file. Besides, in our experiments, we found that define first page of main body is critical. In our future work, we will refine the function about locating this value. Our approach for format checking is using template matching, however, there are still a lot improvement we can do for more precisely checking. [7] introduces to deal with PDF documents by machine learning. For example, CRF tagging (Conditional Random Field) technique, which can define parts of a page using concept of POS (Part-Of-Speech tagging). With CRF tagging, it should be able to provide us a more accurately format information.

REFERENCES

[1] Adobe Acrobat, *What is PDF?* [Online]. Available: https://acrobat.adobe.com/us/en/why-adobe/about-adobe-pdf.html . [Accessed: 10 Mar- 2017].

[2] California State Polytechnic University- Pomona, University Library, *Formatting - Master's Thesis/Project* [Online]. Available: http://www.cpp.edu/~academic-programs/graduate-studies/project-thesis-guidelines/formatting.shtml . [Accessed: 10 Mar- 2017].

[3] PDFBox [Online]. Available: https://pdfbox.apache.org/1.8/cookbook/documentcreation.html. [Accessed: 10 Mar- 2017].

[4] Simone Marinai, Emanuele Marino, Giovanni Soda, *Table of Contents Recognition for Converting PDF Documents in E-book Formats*, DocEng2010, 2010.

[5] Zhaohui Wu, Prasenjit Mitra, C. Lee Giles, *Table of Contents Recognition and Extraction for Heterogeneous Book Documents*, International Conference on Document Analysis and Recognition, 2013.

[6] Geoffrey M. Voelker, *Banal: Because Format Checking Is So Trite*, WOWCS '08, 2008.

[7] Alan Souza, Viviane Moreira, Carlos Heuser, *ARTIC: Metadata Extraction from Scientific Papers using a Two-Layer CRF Model*, DocEng'14, 2014.

[8] Tamir Hassan, *Object-Level Document Analysis of PDF Files*, DocEng'09, 2009.

[9] Øyvind Raddum Berg, *High precision text extraction from PDF documents*,

[10] Xiaojin Zhu, *Conditional Random Fields*, 2010.

[11] Nuno Moniz, Fátima Rodrigues, *Extracting Structure, Text and Entities from PDF Documents*

of the Portuguese Legislation, International Conference on Knowledge Discovery and Information Retrieval, 2012.

[12] IEEE Robotics & Automation Society,

PDF Test [Online]. Available: https://ras.papercept.net/conferences/scripts/pdftest.pl. [Accessed: 21 Mar-2017].

[13] Bruno Lowagie, *iText 7: Building Blocks*, iText Software, 2016.

[14] LATEX-Tutorial.com, *Structuring your document - Adding paragraphs and sections*, [Online]. Available: https://www.latex-tutorial.com/tutorials/beginners/latex-sections/. [Accessed: 21 Mar-2017].

[15] IDRSolutions, JPDF2HTML5, [Online]. Available: https://www.idrsolutions.com/jpdf2html5/. [Accessed: 21 Mar-2017].