PlotExt
Report - Team 6

Contents

1 Installation Guide **4**
- 1.1 System Requirements . . . 4
- 1.2 Linux Installation . . . 5
  - 1.2.1 Installing Python . . . 5
  - 1.2.2 Installing the software from PlotExtOneClick . . . 5

2 User Manual **6**
- 2.1 Functions And their usage . . . 6
  - 2.1.1 Giving PDF as Input . . . 6
  - 2.1.2 Selection of Graphs . . . 6
  - 2.1.3 Generating Output . . . 8
  - 2.1.4 Saving Output in a PDF . . . 11

3 Software Requirements and Specifications **12**
- 3.1 Introduction . . . 12
  - 3.1.1 Purpose . . . 12
  - 3.1.2 Product Scope . . . 12
  - 3.1.3 References . . . 12
  - 3.1.4 Definitions, Acronyms and Abbreviations Used . . . 12
- 3.2 Overall Description . . . 13
  - 3.2.1 Product Perspective . . . 13
  - 3.2.2 Product Functions . . . 13
  - 3.2.3 Operating Environment . . . 13
  - 3.2.4 Design and Implementation Constraints . . . 14
  - 3.2.5 User Documentation . . . 14
  - 3.2.6 Assumption and Dependencies . . . 14
- 3.3 External Interface Requirements . . . 14
  - 3.3.1 User Interfaces . . . 14
  - 3.3.2 Hardware Interfaces . . . 14
  - 3.3.3 Software Interfaces . . . 14
  - 3.3.4 Communication Interfaces . . . 14
  - 3.3.5 Memory Constraints . . . 14
- 3.4 Software Requirements . . . 15
  - 3.4.1 Functional Requirements . . . 15
  - 3.4.2 Performance Requirements . . . 15
  - 3.4.3 Security and Privacy Requirements . . . 15
  - 3.4.4 Reliability . . . 15
- 3.5 Technologies and Languages used . . . 15

4 Description of Algorithms used **16**
4.1 Introduction . . . 16
4.2 Background and Related Work . . . 17
4.3 Graphs Extraction from the pdf . . . 17
4.4 Scale Detection of the graph . . . 18
4.5 Legend Detection . . . 20
4.6 Getting different color plot projections from an image . . . 22
4.7 Getting the table values . . . 23

5 Modules and Architecture **26**
5.1 Modules . . . 26
5.1.1 PDF to Image . . . 26
5.1.2 Graph Extraction . . . 26
5.1.3 Axis Detection . . . 26
5.1.4 OCR Engine . . . 26
5.1.5 Scale Calculation . . . 27
5.1.6 Legend Detection . . . 27
5.1.7 Plot Line Detection . . . 27
5.1.8 Value Calculation . . . 27
5.1.9 Values to PDF . . . 28
5.2 Module Interaction . . . 28
5.3 Software Architecture . . . 29

6 Test Plan **30**
6.1 Introduction . . . 30
6.2 Test Objective . . . 30
6.3 Process Overview . . . 30
6.4 Testing Process . . . 31
6.5 Testing Strategy . . . 31
6.5.1 Unit Testing . . . 31
6.5.2 Integration Testing . . . 32
6.6 Entry and Exit Criteria . . . 34
6.6.1 Unit Testing . . . 34
6.6.2 Integration Testing . . . 36
6.6.3 System Testing . . . 37
6.7 Sample Test Cases . . . 37
6.8 Deliverables . . . 38

1 Installation Guide

The PlotExt Installation Guide describes how to install, uninstall and configure PlotExt on different systems. The guide is primarily written for users and administrators who are familiar with the systems they are working with.The installation guide contains the installation instruction of the softwares and libraries that are to be pre-installed before installing the PlotExt software in your system.

1.1 System Requirements

Make sure that your system conforms to the following requirements to successfully install the software.

- Platform: Linux (Ubuntu 13.04 or higher)
- 256 MB RAM (512 MB recommended)
- Python
- Pyqt4
- numpy
- matplotlib
- opencv
- Imaging
- BeautifulSoup
- QDarksStyle
- reportlab
- Pillow
- pyPdf
- wand
- tesseract-ocr
- python-magick

1.2 Linux Installation

This chapter helps you install the software on the Linux platform using the native Linux Installer.

1.2.1 Installing Python

First,install some dependencies using the following commands:

- **sudo apt-get install build-essential**
- **sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev**

Then download python using the following command:

- **cd /Downloads/ wget http://python.org/ftp/python/2.7.5/Python-2.7.5.tgz**

Extract and go to the directory:

- **tar -xvf Python-2.7.5.tgz**
- **cd Python-2.7.5**

Now,install it using the following command:

- **./configure**
- **make**
- **sudo make install**

1.2.2 Installing the software from PlotExtOneClick

Run the following commands and the software will be installed on the system:

- **python run_install.py**
- **python plotExt.py**

2 User Manual

This software takes a PDF containing graphs as input, and returns an output pdf containing 2-D plot tables for the graphs present in the PDF input. The manual process being quite tedious, automation of the same utilizing this software can result in a phenomenal increase in efficiency.

2.1 Functions And their usage

This sections describes the details for the usage of different functions that the application will be able to perform.

2.1.1 Giving PDF as Input

First step is to load a PDF in the application for which tables are to be obtained. This can be done by clicking on the **Select PDF** button present at left bottom of the UI.

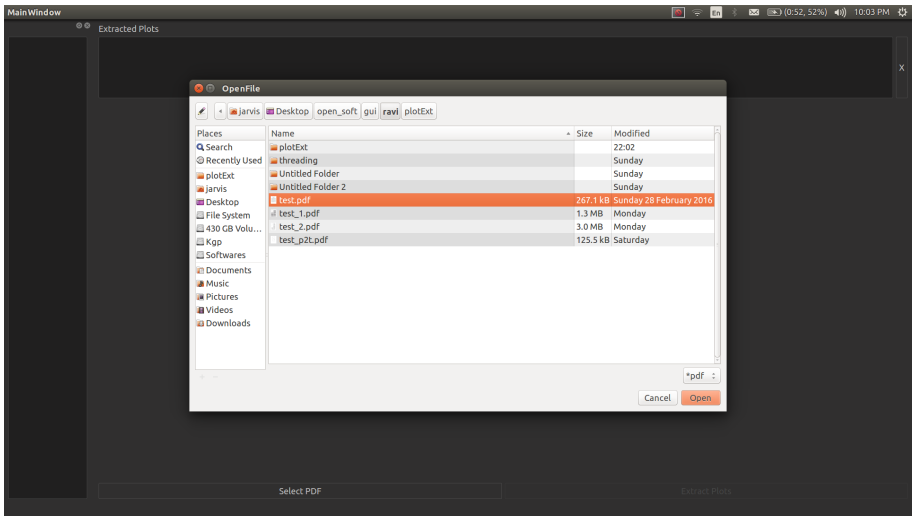


Figure 2.1: Giving PDF as Input.

An Open File dialog will appear after clicking the Select PDF button, enabling the selection of a PDF from any directory by browsing and click on **Open** button after selecting it. The PDF File will be loaded into the main GUI.

2.1.2 Selection of Graphs

This section demonstrates the ways in which the graphs can be selected for processing.

Automatic Detection of Graphs

To detect all the graphs in the selected PDF file after loading it into the GUI, just click on **Extract Plots** Button available at the bottom right of the main GUI.

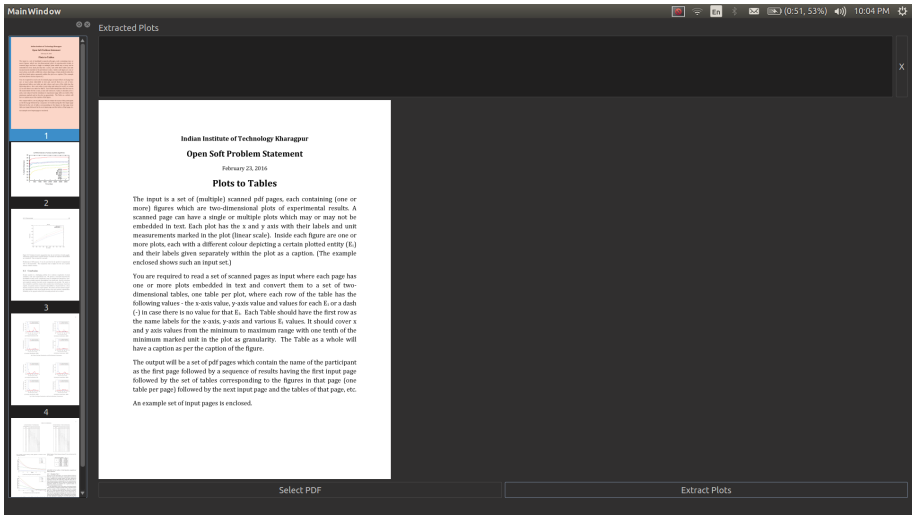


Figure 2.2: Automatic Detection of graphs.

This will automatically detect all the graphs in the given PDF file and display them at the top section of the main GUI.

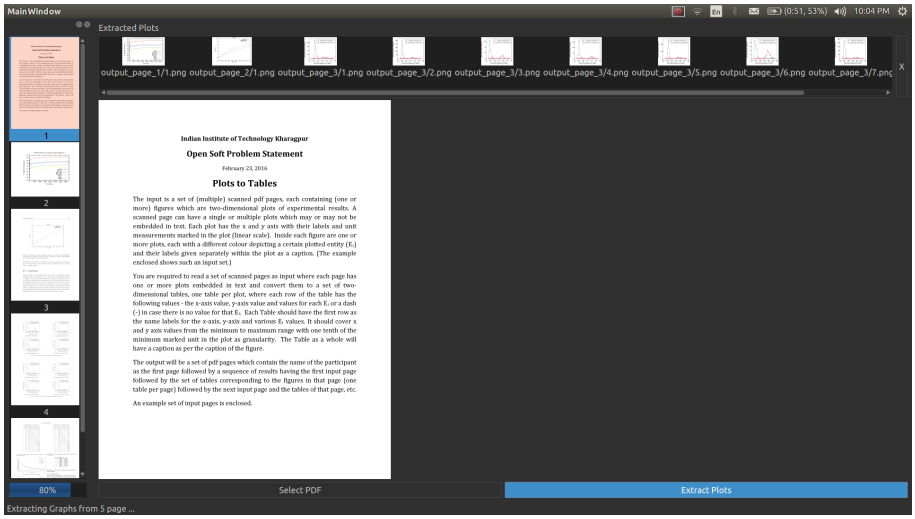


Figure 2.3: Displaying automatically detected graphs at the top.

Deleting Unwanted Graphs

There might be some extra images or unwanted graphs detected, to remove them, just select them one by one and click on the **X** button at the top right section of the Main GUI.

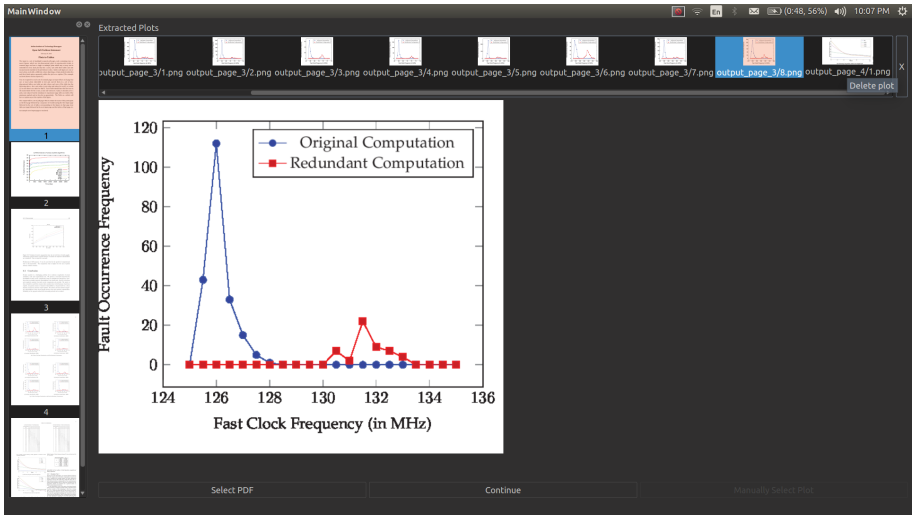


Figure 2.4: Deleting Unwanted Graphs.

Manually Adding Undetected Plots

If there are some plots in the PDF which are not detected, user can manually add them. To do this, first select the PDF Page from the left section which contains the plot and then click on **Manually Select Plot** Button at the bottom. Then select the area of the undetected plot, this will append the plot in the list of plots displayed at the top.

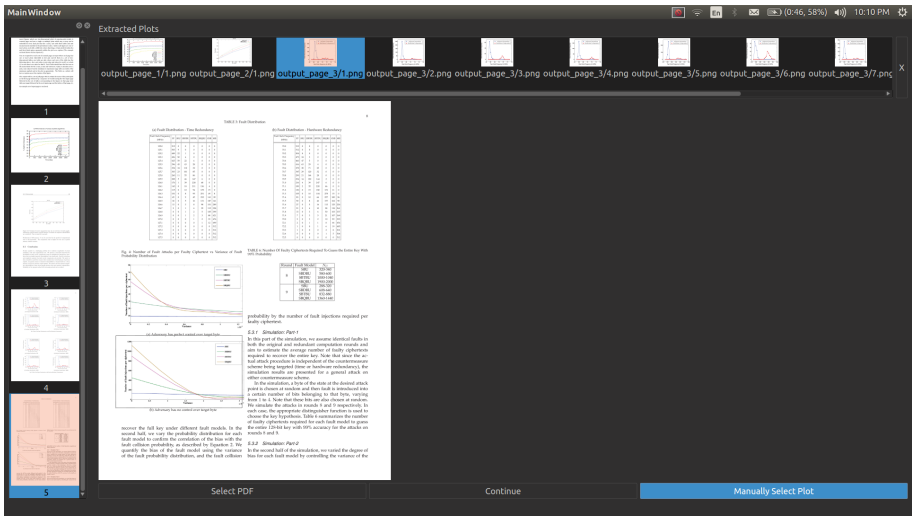


Figure 2.5: Manually Adding Undetected Plots.

2.1.3 Generating Output

Automatically Generating the output

Finally, to generate the output automatically, just click on the **Continue** Button at the Bottom Part of the Main GUI. This will generate the Tables for each of the graph image at the top section of the GUI and display them at the right section.

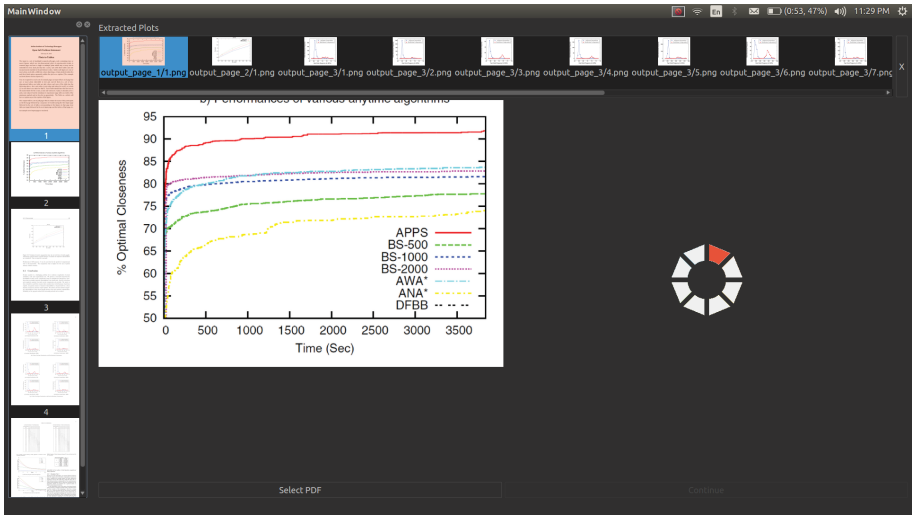


Figure 2.6: Generating Table Values after clicking Continue Button.

Manually Generating the output

Sometimes the software may fail to detect the legend of a required graph, then an error window pops up asking to input manually. Similarly, user can manually give input for a particular plot if he finds the output irrelevant.
So, for the manual scale detection, following steps are to be followed:

1. click on **X1** button at the right bottom section, then
2. click on a point on the X-axis whose (value is known) in the image and
3. then enter this value manually in the textbox just after the **X1** button.
4. Steps 1-3 is to be repeated for **X2** also. From these two distinct points on the axis, its scale can be calculated.
5. Similar process (steps 1-4) is to be repeated for detecting Y-axis
6. Now enter the value for **Number of colors** i.e number of plots in the graph
7. click on the **Proceed** button. This will generate the plot for the graph.

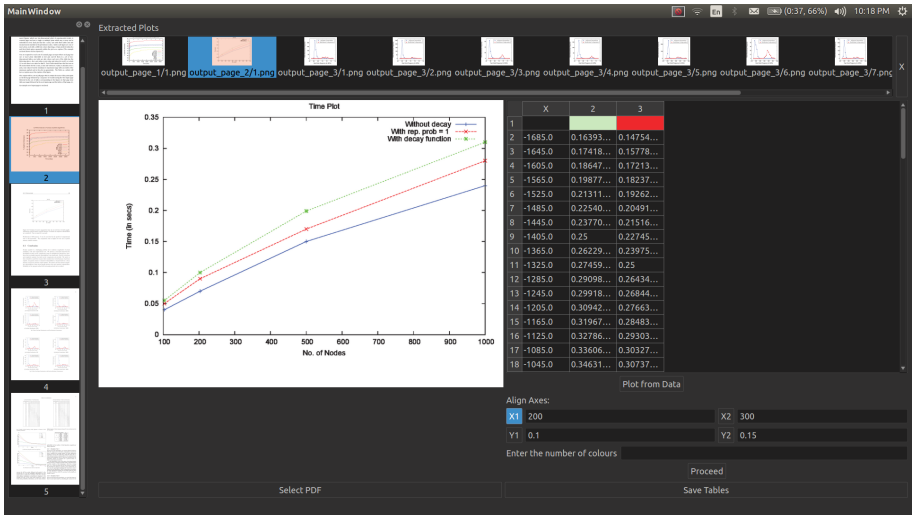


Figure 2.7: Manually Generating the output.

Verification of the output

To verify the output, click on the **Plot Table** button at the bottom right section. This will plot the graph from the generated output table, which can be compared with the input plot for verification.

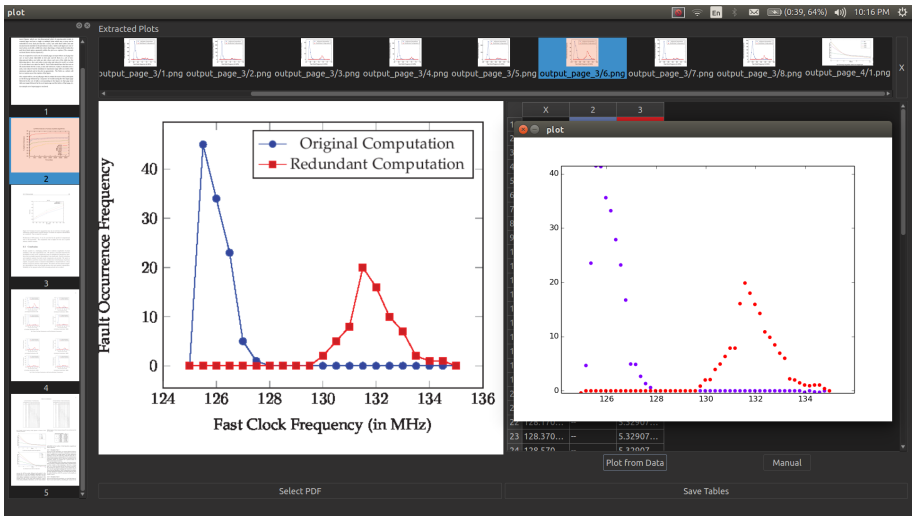


Figure 2.8: Output displayed for Verification.

2.1.4 Saving Output in a PDF

To save the Output in a PDF file, just click on the **Save Table** Button at the bottom right section of the Main GUI, after generating the output tables in the GUI.

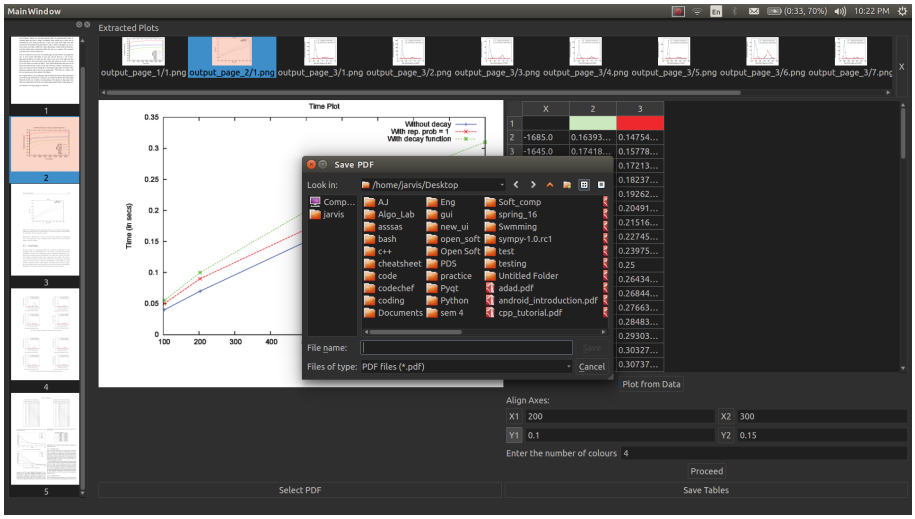


Figure 2.9: Saving output in PDF.

3 Software Requirements and Specifications

3.1 Introduction

3.1.1 Purpose

This SRS describes the software functional and non-functional requirements for release 1.0 of PlotExt software. This software is a standalone application and designed to identify and convert plots to data tables. Unless otherwise stated, all requirements specified here are high priority and committed for release 1.0.

3.1.2 Product Scope

This software consists of the following major functions:

1. Extraction of plots from documents (typed/scanned)
2. Identification axis, scale and legend of the plots extracted
3. Identification multiple plots of different colors.
4. Conversion of plots to data points table and saving it in a pdf.
5. Verification of the extracted data by replotting the data points

3.1.3 References

The document has been prepared according to IEEE STD 830-1998 - Recommended Practices for Software Requirement Specification

3.1.4 Definitions, Acronyms and Abbreviations Used

Table: Table of data points obtained from plots.
Plots: Graphs from research papers
Image: Image mainly corresponds to image of plot
GUI: Graphical User Interface
OCR: Optical Character Recognition

3.2 Overall Description

3.2.1 Product Perspective

This is a software program that converts plots to tables. It also generate tables for multiple plots in the same figure assuming individual plots are plotted with different colors. The results are shown on the screen and can also be saved in the form of pdf. The data points can be replotted within the software to validate predictions. This software is a standalone application and does not require internet connection.

3.2.2 Product Functions

Select PDF
User will select the PDF to be processed(to get data from plots in the PDF) by using a file dialog box.

Run
Plots are extracted from the PDF and shown to the user using this function.

Deleting extracted plot
If some images are extracted wrongly as graphs or if he doesnt required data from a particular graph, the user can remove them manually by deleting the images from the plot list.

Automatic
The data is generated from the plot using automatic mode which uses the internal axis detection, legend detection modules without any user input. The final tables are generated using the internal modules to get the plot information.

Manual
If the user is not satisfied with the automatic output, he can input information such as scale, number of plotlines, selecting area of plot to extract plot from the pdf etc. such that the data is extracted from the plot using user given information. This is mainly used in the case of bad quality images and/or when OCR engine doesnt obtain the required text.

Save
The obtained data is saved in a PDF document with a plot in each page followed by the corresponding data table.

3.2.3 Operating Environment

This software is developed in python and it is compatible with Windows as well as Ubuntu operating system. It can be installed on Windows 7 and higher and Ubuntu 12.04 and higher. For windows platform, py2exe must be installed on the system.

3.2.4 Design and Implementation Constraints

OCR functions provided by Tesseract [10] are used for scale and legend identification. If the characters are not recognized by OCR user input may be needed for scale, boundary and legend.

3.2.5 User Documentation

A user manual and installation guide in pdf format would be made available along with the software.

3.2.6 Assumption and Dependencies

This software being developed in python and cannot be installed if python environment is not available.

3.3 External Interface Requirements

3.3.1 User Interfaces

The user interface should be simplistic. It should contain vertical bar to display different pages in pdf, horizontal bar to display extracted plots from pdf. In the central region, there should be plot with its datatable on left side.

3.3.2 Hardware Interfaces

No special hardware is needed for the functioning of this software. A computer with a monitor, a keyboard and a mouse suffices.

3.3.3 Software Interfaces

The software consists of a single user system. It does not require any internet connection. The inputs are given in pdf format which needs to be parsed and the result is also stored in the pdf format. Different libraries including openCV, PyQt, etc. needs to be installed on the system before running this application.

3.3.4 Communication Interfaces

Since this is a standalone application and does not require any internet connection, no communication interface is required for this application.

3.3.5 Memory Constraints

Secondary memory storage is required to store the input files and the result produced as output by the application. Sufficient primary memory storage should be available for the smooth running of the software, though the software does not require very high primary memory storage.

3.4 Software Requirements

3.4.1 Functional Requirements

1. PDF files needs to be converted to images. Each plot from those pages needs to identified and converted to images.
2. Axis detection should be performed on plot images where scale needs to be identified.
3. This part should detect the colour of the various legends and the text corresponding to each legend. It should also remove the legend part from the graph.
4. Scale of plots should be identified in order to get data points.
5. Identified values should be outputted in table format. Further, it should convert those tables in PDF.

3.4.2 Performance Requirements

After giving a pdf as input and running the application it takes few seconds to generate the result containing pdf. No error or crash cases were detected during the runtime of the application.

3.4.3 Security and Privacy Requirements

The software does not require submission of sensitive information, and hence no security and privacy requirements have been identified.

3.4.4 Reliability

The software depends on the number of graphs present in the given input pdf. Hence its accuracy of results depends on the processing speed of the system and the complexity of plots present in the input pdf. No cases of getting an error or a crash were identified.

3.5 Technologies and Languages used

1. OpenCV
2. Python
3. Tesseract OCR engine

4 Description of Algorithms used

Abstract

Finding data points from plots is important task in analyzing, studying and verifying plots. This can be very useful in detecting falsification of data, dropping data points and result manipulations. It is found that, on average, about 2% of scientists admitted to have fabricated, falsified or modified data or results at least once a serious form of misconduct and up to one third admitted a variety of other questionable research practices including dropping data points based on a gut feeling, and changing the design, methodology or results of a study in response to pressures from a funding source[1]. Jan Hendrik Schn (physics of semiconductors), forged results, using the same graph image in different contexts[2]. Apart from verifying data, it can be used to obtain data from scientific experiments when only the graph information is available. Our heuristics identifies plots which may or may not be embedded in texts and converting them to tables. Here we present a novel approach for detecting values of multiple line plots plotted using different colors using the information from legends within the plot.

4.1 Introduction

Plots are integral part of research literature. Often researchers provide just the graphs of experiments in scientific papers while not making the underlying data accessible. This makes it hard to examine or use the data in a more critical perspective. Providing a tool to extract data from graphs will give valuable insight into the experiments presented in scientific papers and as well help in validating reproducibility of the experiments. Thus such a functional plot to data tool will be of valuable use in the scientific community.Although 2-D plots are easily understood by human users, current search engines rarely utilize the information contained in the plots to enhance the results returned in response to queries posed by end-users. Automatic data extraction from 2-D plots is challenging due to diversity among figures, various imaging conditions, and the presence of noise. Furthermore, to extract a line in a figure accurately, the tool must determine the continuation of each line beyond cross-over points (i.e., points where two lines intersect). There exist semi-automatic tools for extracting data from 2-D plots. These tools may provide interactive interfaces for users to input requests and update results of automated extraction. For instance, users can select a data point or a line and the tools will generate a list of corresponding data values. The semi-automatic tools assist users to process individual plots. However, the amount of human interaction necessary for every single plot makes it unusable for processing a large volume of figures in a digital library.Users can analyze the data easily with the aid of scalable automated data extraction tools. The data generated by these tools may serve as raw data for a variety of analyses, e.g., curve fitting of the raw

data, analysis of published models of scientific phenomena, interpretation,and predictions.

4.2 Background and Related Work

Recognition of special types of graphics, especially engineering graphs, has attracted a lot of attention. Terrades et al. [4] presented a random transform based method to represent symbol graphics. Henderson et al. [5] proposed a structural modeling approach with a non deterministic agent system to interpret scanned CAD drawings. Blostein et al. [3] surveyed a selection of techniques for diagram recognition. Lank et al. [6] proposed an interactive diagram recognition method. Luo et al. [8] proposed a framework for engineering drawings recognition using a case-based approach. Lu et al. [7] developed a figure categorization system. Zhou et al. [9] proposed a hough transform based technique for chart recognition. The problem of extracting data from lines in 2-D plots has not been addressed specifically.

4.3 Graphs Extraction from the pdf

To extract all the graphs from a pdf , first convert the pdf pages of the pdf to images. Firstly, a blank image has been created and required density is assigned to it. When the pdf page is read using this image file, it reads in the specified density. After that all pdf pages are written in image format.

After converting the pdf file to images all the graphs are extracted from each image file. At first, it does the probabilistic Hough transform on the image given and the image file is read and converted to a gray-scale image. After that Adaptive Thresholding is done, which converts gray-scale image to binary one, followed by morphological transformation. And then Image blurring/smoothing is done followed by straight lines detection.

Algorithm 4.3.1: HOUHGP($page$)

$g \leftarrow MorphologyExtraction(adaptiveThreshold(gray(page))$
$lines \leftarrow HoughLinesP(g)$
$im2 \leftarrow blackblankimage$
for $i \leftarrow 1$ **to** $len(lines)$
  do plot lines on im2
return $(im2)$

After the Hough Transform is done to approximate the shape on the contours detected and store them in a list. Now for each contour detected in this list contour area is calculated and is compared with the threshold area. If this area is greater than the threshold area, then the shape of the contour is approximated. If the length of this approximated shape is between two and ten, the coordinates of the bounding rectangle of this contour is found. After doing this the coordinates of the graph were found and this portion of the image is stored in a array and all such graphs were appended to this array. Fig 4.1 shows

an example of an image of a graph extracted from the input PDF.

Algorithm 4.3.2: POLYDP($im2$)

$cnts1 \leftarrow \emptyset$
$temp \leftarrow findContours(gray(im2))$
for $i \leftarrow 1$ **to** $len(temp)$
  do { **if** $contourArea(temp[i]) > thresholdArea$
    then Add temp[i] to cnts1
for $i \leftarrow 1$ **to** $len(cnts1)$
  do { **for** $j \leftarrow i+1$ **to** $len(cnts1)$
    do { Check for overlap of cnts1[i] and cnts1[j]
      Merge using convex hull if overlapped
$cnts \leftarrow$Merged list of the contours
$x \leftarrow \emptyset, y \leftarrow \emptyset, w \leftarrow \emptyset, h \leftarrow \emptyset$
for $i \leftarrow 1$ **to** $len(cnts)$
  do $x[i], y[i], w[i], h[i] \leftarrow boundingRect(cnts[i])$
for $i \leftarrow 1$ **to** $len(x)$
  do { **for** $j \leftarrow 1$ **to** $len(x)$
    do { Compare x[i],x[j] and y[i],y[j] to find outer bounding boxes
$list1 \leftarrow \emptyset, list2 \leftarrow \emptyset, list3 \leftarrow \emptyset$
for each outer bounding boxes
  do { Add cropped image of plot with labels to list1
    Add name of plot to list2
    x and y axis parameters of the bounding box tolist3
return $(list1, list2, list3)$

Algorithm 4.3.3: GRAPHEXTRACT($pagelist$)

/* pagelist is the list of the page images of the input document*/
$numPages \leftarrow len(pagelist)$
for $i \leftarrow 1$ **to** $numPages$
  do { $page \leftarrow pagelist[i]$
    $ga, na, ia \leftarrow polydp(houghp(page))$
    Return the lists to the main thread and continue

4.4 Scale Detection of the graph

For the scale detection part at first the axis of the graph was determined and then its coordinates were used to extract out the scales. This determination of the coordinates of the axis was done via the hough transform algorithm. Now once the axis was determined the image containing the graph was cropped below the coordinates of the lower x axis and then the OCR was done on this cropped image to get out the x-scale values.

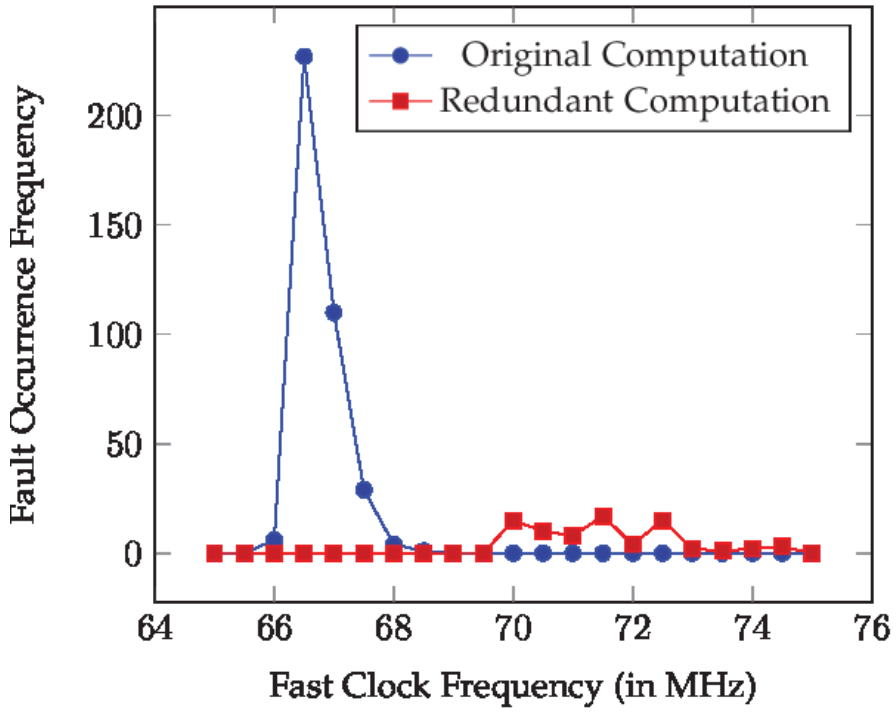


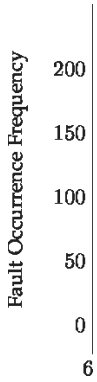
Figure 4.1: Image of a graph extracted from the pdf

Algorithm 4.4.1: FINDAXIS($ImageofPlotwithlabels$)

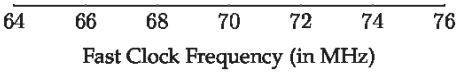
$g \leftarrow morphologyExtraction(adaptiveThreshold(gray(image)))$
$lines \leftarrow HoughTransformP(g)$
$im2 \leftarrow$ black blank image
for i $\leftarrow$ 1 **to** $len(lines)$
  do {plot the lines after houghtransform on im2
$cnts \leftarrow findContours(im2)$
for i $\leftarrow$ 1 **to** $len(cnts)$
  do $\{x[i], y[i], w[i], h[i] \leftarrow boundingRectangle(cnts[i])$
for i $\leftarrow$ 1 **to** $len(cnts)$
  do $\begin{cases} for \text{ j} \leftarrow 1 \textbf{ to } len(cnts) \\ \quad \textbf{do } \{\text{compare x[i],y[i],w[i],h[i] with x[j],y[j],w[j],h[j] to-} \\ \quad \text{find outer contour} \end{cases}$
return (Parameters of the bounding box)
/* this bounding box contains only the graph or plot */

For the extraction of the y-scale,as done previously the original image was cropped such that the new image contained only the region left to the leftmost axis.Now the OCR could not be done directly on this cropped image as it contains texts to the left of the scale.So

to crop out the texts the image was first rotated by 90 degrees in the clockwise direction. Now on this rotated image OCR was run to determine the coordinates of the text.This text region was then cropped out of the image and then this image was rotated again by 90 degrees in the clockwise direction.Optical Character Recognition was finally done on this image to return the corresponding y-scale values of the graph. Figure 4.2 shows an example of x-axis and y-axis scale being detected for the figure 4.1.



(a) y-axis scale



(b) x-axis scale

Figure 4.2: Scale detection of a graph

4.5 Legend Detection

This part involves detecting the color of the various legends and the text corresponding to each legend. This algorithm removes the legend part from the graph as it may create a problem later as we may confuse it with the real graph. Also, note that, the output contains bounding box of the text and not the original text, because we cannot rely on OCR always for detection of text. So, even if the OCR detects one of the texts, this code detects all others and finally gives them as the output. The text can later be retrieved from the bounding boxes.

Algorithm 4.5.1: LEGEND_DETECT($image, boundingBoxParamters[]$)

$ocrDetection(image)$/* returns the hOCR file */
$parseHocr(hOCR_file)$
$extractTextAndRectangles(hOcrFile)$
$ignoreSomeText()$
/* we now have the text and bounding box of rectangle around it */
$removeTextOutsideBoundingBox()$
if $(r[0] > xMin$ **and** $r[1] > yMin$ **and** $r[2] < xMax$ **and** $r[3] < yMax)$
then $\{retain(r)$ /* r contains diagonal endpoints */
$mergeRectInSameLine()$
$createSetsOfElementsInSameHorizontalLevel()$
/*sets of all rectangles which need to be merged*/
$formABigRectangleOutOfSmallRectangle(set)$
/*text corresponding to each legend in one box*/
$moveUpAndDown()$
while $(detectBlackDensity()andinsideRange())$
do $\begin{cases} insert(rect) \\ moveUp() \end{cases}$
while $(detectBlackDensity()andinsideRange())$
do $\begin{cases} insert(rect) \\ moveDown() \end{cases}$
$findLegendColorAndExtent()$
for each rect
do $\begin{cases} searchForDirectionOfFirstColouredPixel() \\ storeTheColor(pixel) \\ \textbf{if} \text{ left} \\ \quad \textbf{then} \begin{cases} \textbf{while } legendExists() \\ \quad \textbf{do } \{moveLeft() \;\; storeTheBoundaries(legend) \end{cases} \\ \quad \textbf{else} \begin{cases} \textbf{while } legendExists() \\ \quad \textbf{do } \{moveRight() \;\; storeTheBoundaries(legend) \end{cases} \end{cases}$
$findBox(legendRect, textRect)$ /*box which just contains legends and text*/
$image.erase(box)$ /*makes the above found box white*/
$image.erase(box)$ /*text and legend from the image are removed*/

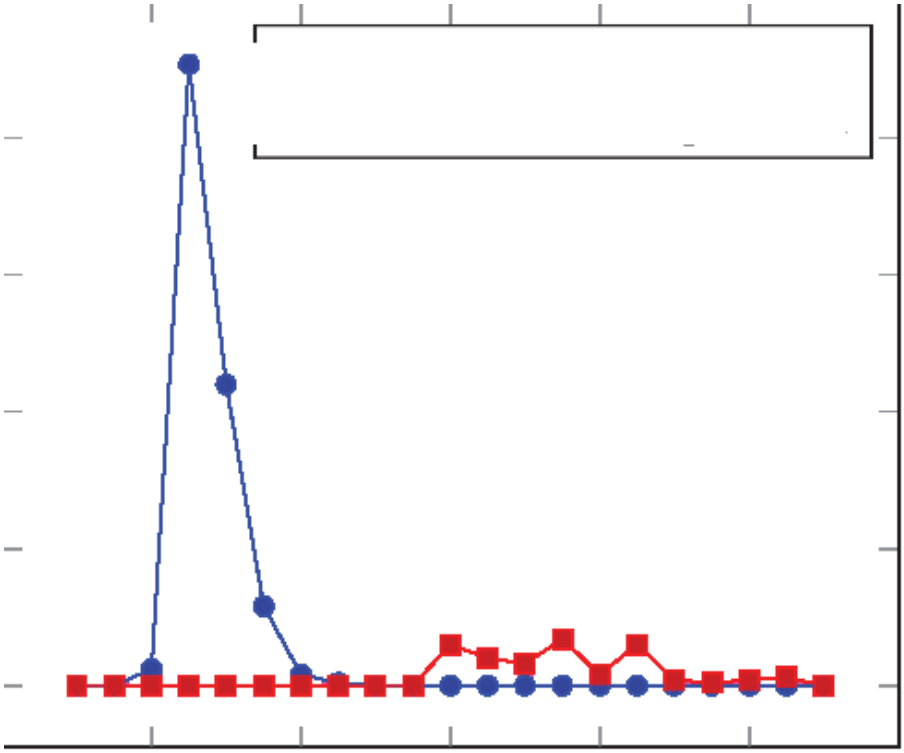


Figure 4.3: Image without legend

4.6 Getting different color plot projections from an image

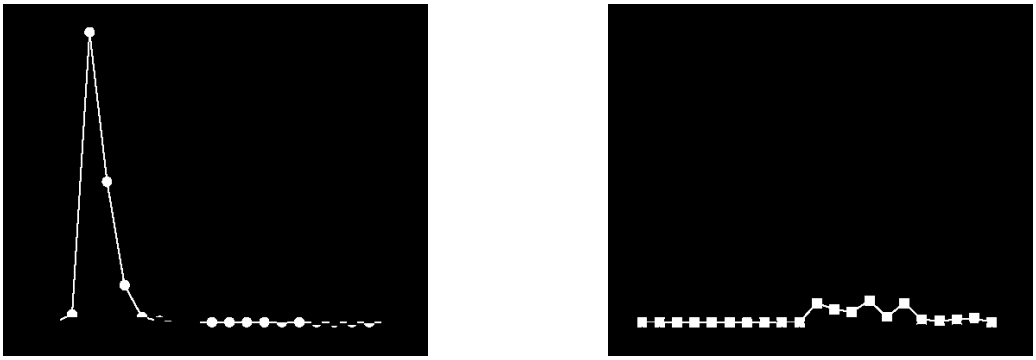
After the legend detection part, we get list of (b,g,r) values of the legend colors. The list of (b,g,r) values of each legend is taken and converted to HSV(Hue Saturation Value) format. Then for each of these colors, the entire image is traversed and pixels having hue and saturation values within a particular threshold are marked and counted. If any of these colors occurs for more than 75 percent of the total image area then those points are considered as background and rejected. After this a binary image is generated based on every color and are stored in a list. This list contains binary images each of which correspond to a particular color. Figure 4.3 shows an example of two graph images that were obtained from the Figure 4.1.

Algorithm 4.6.1: MARKCOLOURS($image, hue$)

$a \leftarrow \emptyset$
$im \leftarrow$blank black image
for each $pixelsinimage$
  do { **if** hue(pixel) is within hue threshold
    then Add pixel to a
for $i \leftarrow 0$ **to** $len(a)$
  do mark im(a[i]) white
return (im)

Algorithm 4.6.2: FINDCOLOURSMANUA($image, k$)

```
/*k ← numberofclusters*/
imageList ← ∅
hueHistogram ← ∅
hueList ← ∅
hueHistogram ←sorted values of hues based on number of pixels of that hue
hueList ←append first k hue values that are non overlapping
/*Overlapisdecidedbasedonathreshold*/
for i ← 0 to len(hueList)
  do  add markColours((image,hueList[i])) to imageList
return (imageList)
```



(a) Blue plot (b) Red Plot

Figure 4.4: Getting different color plot projections from a graph

4.7 Getting the table values

In this part, interpolated plots are given as inputs. The parameters are interpolated plot image, number of pixels between two consecutive x-axis divisions, number of pixels between two consecutive y-axis, the length of the bounding box of the graph along the x-axis, the length of the bounding box of the graph along y-axis, the coordinate from where x-axis starts, the coordinate from where y-axis starts, units per division on the x-axis, units per division on the y-axis.

Algorithm 4.7.1: FINDVALUE($binaryImage, x$)

```
a ← ∅
for each i ← 0 to len(binaryImage)
  do { if binaryImage(i,j) is white
     { then a ← a + i
if length(a) = 0
  then return (NULL)
  else return (average(a))
```

A K means clustering algorithm is applied to the graphs and binary color detected images are found out. Each of these images are expected to contain the plot of a single variable. Each of these plots are sent as input to the function which returns the table from the plot of a single variable. The function takes the binary image of the plot as input. We traverse the image along the x-axis, and for a particular x, the value for y=f(x) is found by traversing along the y-axis and finding out the white pixel in that particular column. If there are multiple white pixels for a given x, the value of y=f(x) is the average of those pixel values. If there are no white pixels detected for a particular x, then the function has a discontinuity at that point and a **null** value is returned.

Algorithm 4.7.2: APPROXTABLE($binaryimage, pixelperdivX, pixelperdivY, size(boundingBox), originX, originY, scaleX, scaleY$)

$interval \leftarrow scaleX/10$
$yValue \leftarrow \emptyset$
while j < max value on x-axis
do $p \leftarrow$ pixelValue of j
$v1 \leftarrow findValue(binaryImage, p-1)$
$v2 \leftarrow findValue(binaryImage, p+1)$
if v1 = NULL **and** v2 = NULL
then $yValue \leftarrow$ yValue + NULL
if any one of v1 and v2 is NULL
then $yValue \leftarrow$ yValue + (non NULL value+originY)
else $yValue \leftarrow$ yValue + ((v1+v2/2)+originY)
$j \leftarrow$j + interval
return ($yValue$)

To remove the discontinuities created due to noise or if the graph lines are dotted, line interpolation is used. The line interpolation function joins discontinuities by a straight lines if the length of the discontinuity is below a particular threshold distance. This is done to overcome the problems created if the plot lines are dotted or due to noise.

Algorithm 4.7.3: FINDTABLES($listOfBinaryImages, pixelperdivX, pixelperdivY, size(boundingBox), originX, originY, scaleX, scaleY$)

$tables \leftarrow \emptyset$
for each $allimage \in listOfBinaryImages$
do $tables.append(approxTable(image, pixelperdivX, pixelperdivY, size(boundingBox), startingValueX, startingValueY, scaleX, scaleY))$
return ($tables$)

Algorithm 4.7.4: FINDPARAMETERS($x1, pixelX1, x2, pixelX2, y1, pixelY1, y2, pixelY2, scaleX, scaleY, size(boundingBox)$)

compute the pixelValues and values of originX and originY
compute the pixelperdivX and pixelperdivY
$findTables(listOfBinaryImages, pixelperdivX, pixelperdivY, size(boundingBox), originX, originY, scaleX, scaleY)$

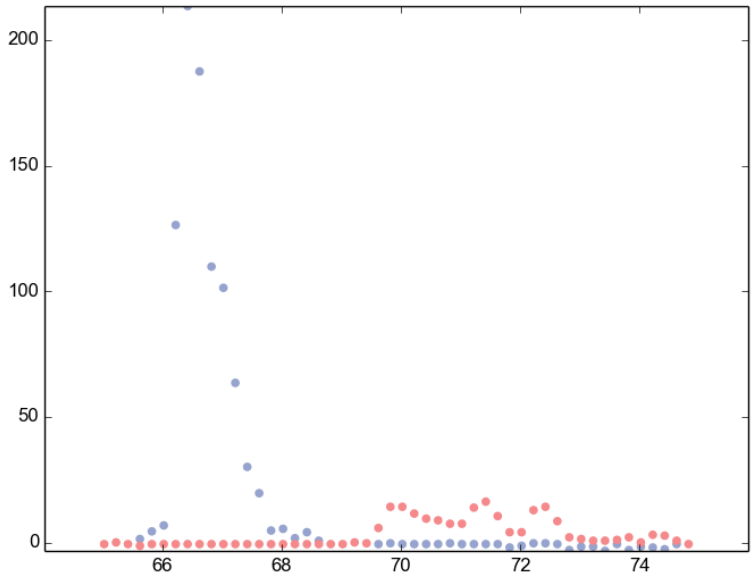


Figure 4.5: Image generated for calculating table values

5 Modules and Architecture

5.1 Modules

5.1.1 PDF to Image

Input : PDF file
Output : Image file
Descriptions : This creates blank image file with required density. PDF page is read using this image file. After that all PDF pages are written in image format.

5.1.2 Graph Extraction

Input : A list of pages of the pdf document in the png format
Output : Three lists –list of images of bounding boxes containing graphs with labels, names of these images as written in file system,and lastly a list containing the parameters of only the bounding box
Descriptions : First the image is converted to gray scale, followed by Gaussian blurring (to remove noise) and then adaptive threshold is applied to it so that extraction from the page image can be more fruitful.Next probabilistic Hough trasnform is applied to it to find out the potential horizontal and vertical lines on the image which can be a part of the bounding box of a graph. After that on the image formed on the application of hough transform , contour detection is applied which clusters the lines if they form some contour. Using different methods like convex hull and area estimation overlapping contours are then merged. Next we use an area threshold to discard smaller contours and we also find the outermost contour in case of nested ones.Also an approximation on aspect ratio is made to make the extraction more fruitful.

5.1.3 Axis Detection

Input : An image of the graph extracted from 5.1.2 above
Output : The bounding box outlining the axis of the plot
Descriptions : A process similar to the previous step is being undertaken.First the houghlines are being plotted and then contour detection is being done. Overlapping contours are merged using a convex hull and then the required bounding box is extracted.

5.1.4 OCR Engine

Input : Image File
Output : Text recognized and the position of the text(s) recognized in the format of HOCR

Descriptions : Tesserac-Ocr package is used as the OCR engine by this software. This engine is used for detecting scale of the axes, legend of the plot lines, title of the plot etc.

5.1.5 Scale Calculation

Input : The graph with its bounding box coordinates known
Output : The x and y scale values of the graph along with the position of markings
Descriptions : First the image was cropped below the x-axis and then the OCR was run on it to extract the x scale values. For the extraction of the y-scale,as done previously the original image was cropped such that the new image contained only the region left to the leftmost axis.Now the OCR could not be done directly on this cropped image as it containes tetxs to the left of the scale.So to crop out the texts the image was first rotated by 90 degrees in the clockwise direction. Now on this rotated image OCR was run to determine the coordinates of the text.This text region was then cropped out of the image and then this image was re-rotated again by 90 degrees in the clockwise direction.OCR was finally done on this image to return the corresponding y-scale values of the graph.

5.1.6 Legend Detection

Input : The cropped image extracted from the pdf and the bounding_box (rectangle containing x-y axes) coordinates of the graph
Output : The image with legend and the corresponding text removed, and an array containing the the bounding_box of the rectangles around the text and the RGB value of the color of the corresponding legend.
Descriptions : This part involves detecting the colour of the various legends and the text corresponding to each legend. This code also removes the legend part from the graph as it may create a problem later as we may confuse it with the real graph. Also, note that, the output of the code contains bounding box of the text and not the original text, because we cannot rely on OCR always for detection of text. So, what this code does is, even if the OCR detects one of the texts, it detects all others and finally gives them as the output. The text can later be retrieved from the bounding boxes.

5.1.7 Plot Line Detection

Input : List of (b,g,r) values of the legend colours
Output : List of binary images each of which correspond to a particular colour (or plot)
Descriptions : The list of (b,g,r) values of each legend is taken as input and converted to HSV format. Then for each of these colours, the entire image is traversed and pixels having hue and saturation values within a particular threshold are marked and counted. If any of these colours occurs for more than 75% of the total image area then those points are considered as background and rejected. After this a binary image is generated based on every colour and are stored in a list. This list of binary images is the output.

5.1.8 Value Calculation

Input : Position(in pixels) and value of two points on x-axis, Position(in pixels) and value of two points on y-axis, Position(in pixels) of bottom left and top right corner of

the graph bounding box, scale along x-axis and y-axis, list of binary images each of which corresponds to one plot in the graph
Output : Tables of all the plots in a graph
Descriptions : The function finds the table of one plot at a time. If there are multiple plots in one graph, the function is called multiple times corresponding to each plot. For a binary image of the plot, to find the value for any x, we traverse along the y direction for that pixel and find the white pixel. If we dont find any white pixel for a particular x, then the plot is assumed to be discontinuous at that x and some null value is returned.

5.1.9 Values to PDF

Input : Location and Name of the PDF File,List of X-Y values, List with location of each plot, List of Title/Caption of each plot, 2-D List having Titles of X and Y axes (detected from the legend and axis itself) of each graph.
Output : A PDF File containing the output as specified.
Descriptions : This part involves giving of output in a well organised PDF form for the selected graphs. If there are multiple plots in a single graph, output is given in a single table having different columns for each plot line.

5.2 Module Interaction

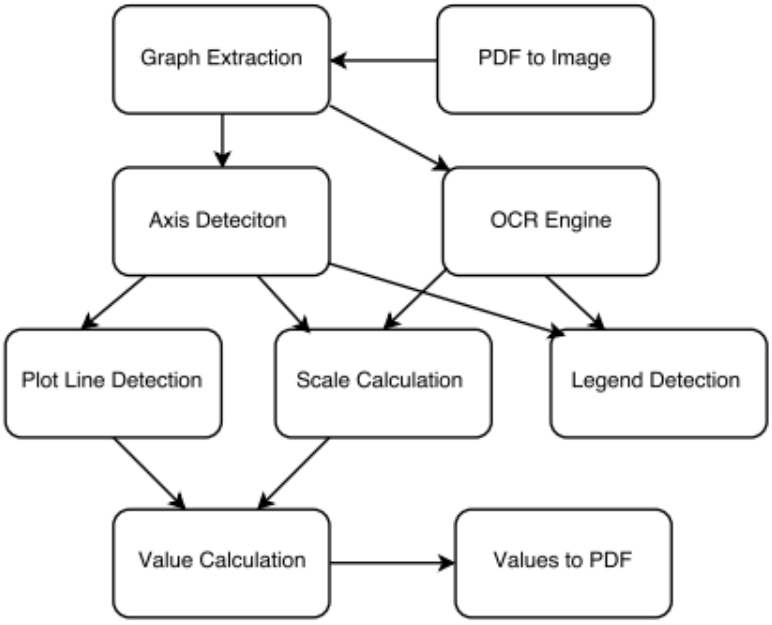


Figure 5.1: Modules Interactions-Data Flow.

5.3 Software Architecture

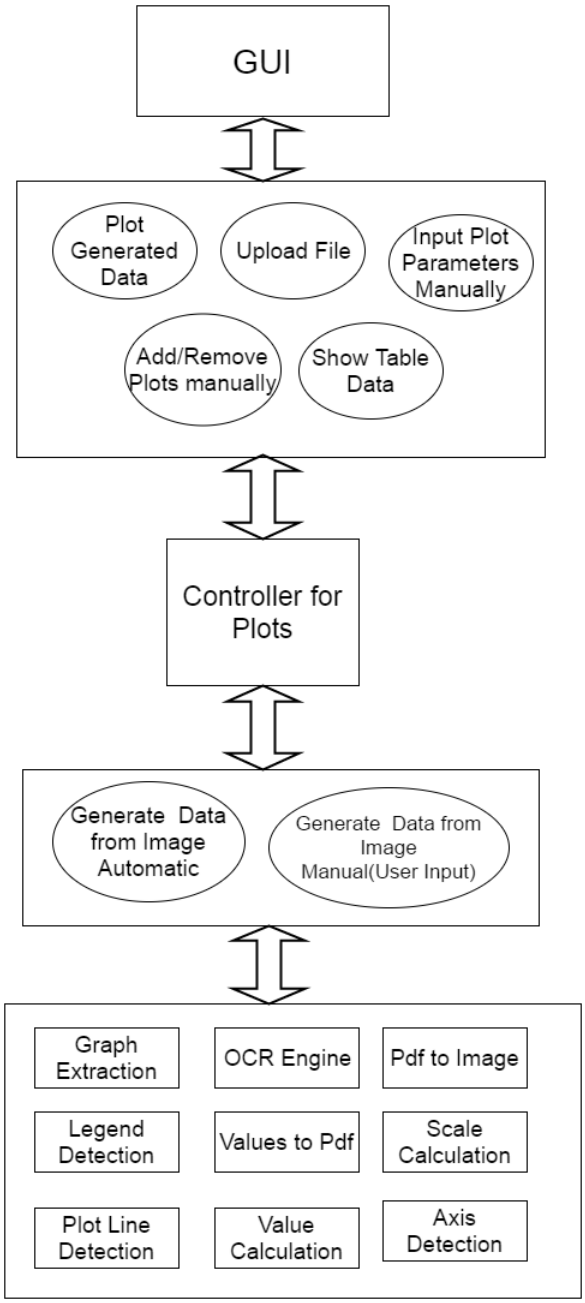


Figure 5.2: Architectural Diagram.

6 Test Plan

6.1 Introduction

Its objective is to communicate project-wide quality standards and procedures. It portrays a snapshot of the project as of the end of the planning phase. This document will address the different standards that will apply to the unit, integration and system testing of the specified application. Testing criteria under the white box, black box, and system-testing paradigm will be utilized. This paradigm will include, but is not limited to, the testing criteria, methods, and test cases of the overall design.

6.2 Test Objective

The objective of this test plan is to find and report as many bugs as possible to improve the integrity of our program. Although exhaustive testing is not possible, a broad range of tests will be exercised to achieve the goal. There will be following functions that can be performed by this application: taking a pdf file as an input, plotting 2-D tables for the graphs present in the input pdf which can either be done automatically or manually, displaying the graphs and their corresponding tables in the application, deleting extracted plots, saving the extracted data in pdf format. The user interface to utilize these functions is designed to be user-friendly and provide easy access to all the functions.

6.3 Process Overview

The following represents the overall flow of the testing process:

1. Identify the requirements to be tested. All test cases shall be derived using the current Program Specification.
2. Identify which particular test(s) will be used to test each module.
3. Review the test data and test cases to ensure that the unit has been thoroughly verified and that the test data and test cases are adequate to verify proper operation of the unit.
4. Identify the expected results for each test.
5. Document the test case configuration, test data, and expected results.
6. Perform the test(s).

7. Document the test data, test cases, and test configuration used during the testing process. This information shall be submitted via the Unit/System Test Report (STR).
8. Successful unit testing is required before the unit is eligible for component integration/system testing.
9. Unsuccessful testing requires a Bug Report Form to be generated. This document shall describe the test case, the problem encountered, its possible cause, and the sequence of events that led to the problem. It shall be used as a basis for later technical analysis.
10. Test documents and reports shall be submitted. Any specifications to be reviewed, revised, or updated shall be handled immediately.

6.4 Testing Process

The diagram above outlines the Test Process approach that will be followed.

1. Organize Project involves creating a System Test Plan, Schedule and Test Approach, and assigning responsibilities.
2. Design/Build System Test involves identifying Test Cycles, Test Cases, Entrance and Exit Criteria, Expected Results, etc. Test Cases and the Data required are identified. The Test conditions are derived from the Software Requirement Specifications.
3. Design/Build Test Procedures includes setting up procedures such as Error Management systems and Status reporting.
4. Build Test Environment includes requesting/building hardware, software and data setups.
5. Execute System Tests - The tests identified in the Design/Build Test Procedures will be executed. All results will be documented and Bug Report Forms filled out and given to the Development Team as necessary.
6. Signoff - Signoff happens when all pre-defined exit criteria have been achieved.

6.5 Testing Strategy

The following outlines the types of testing that were done for unit, integration, and system testing. It includes what will be tested, the specific use cases that determine how the testing is done.

6.5.1 Unit Testing

Unit Testing is done at the source or code level for language-specific programming errors such as bad syntax, logic errors, or to test particular functions or code modules. The unit test cases shall be designed to test the validity of the programs correctness.

White Box Testing

In white box testing, the UI is bypassed. Inputs and outputs are tested directly at the code level and the results are compared against specifications. This form of testing ignores the function of the program under test and will focus only on its code and the structure of that code. Test cases are generated that not only cause each condition to take on all possible values at least once, but that cause each such condition to be executed at least once.

Black Box Testing

Black box testing typically involves running through every possible input to verify that it results in the right outputs using the software as an end-user would. We have decided to perform Error guessing and Boundary Value Analysis testing on our application.

6.5.2 Integration Testing

Incremental Testing

There are 10 primary modules that were needed to be integrated, viz., GUI modules and 9 back-end modules. Integration of back-end modules were tested in white box testing as well as during software development and hence are less prone to errors in Incremental testing. These components, once integrated form the complete PlotExt application. The following describes these modules as well as the steps that will need to be taken to achieve complete integration. We will be employing an incremental testing strategy to complete the integration.

Module-1: Graphic User Interface (GUI) Module. This module provides a simple GUI where the user can perform the different actions (functions). This module was tested separate from the backend to check if each interface is functioning properly, and in general, to test if the mouse-event actions were working properly. The testing was performed by writing a GUI testing for each element in the interface.

Module-2: Back-end Modules. Back-end Modules contains modules such as PDF to image, Graph Extraction, Axis Detection, Scale Calculation, Legend Detection, Plot Line Detection, Value Calculation, Values to PDF. All these modules were integrated with each other and while integration testing was performed on taking into consideration of all input and output of modules. When integration of above modules are integrated with GUI modules, we get complete application. Before integrating them all elements of GUI were tested for calling appropriate functions and their input and output with GUI.

System Testing

The goals of system testing are to detect faults that can only be exposed by testing the entire integrated system or some major part of it. Generally, system testing is mainly concerned with areas such as performance, security, validation, load/stress, and configuration sensitivity. But in our case we focused only on function validation and performance. And in both cases we used the black-box method of testing.

Function Validation Testing

The integrated PlotExt Application was tested based on the requirements to ensure that we built the right application. In doing this test, we tried to find the errors in the inputs and outputs, that is, we tested each function to ensure that it properly implements the parsing procedures, and that the results are expected. We tested the following functions:

- **Select PDF** It was checked that whether the user is able to select a pdf to be given as input by using a file dialog box.
- **Run** Plots are extracted from the input pdf and are shown to the user using this function.
- **Deleting Extracted Plot** If some images are wrongly extracted as graphs or the extracted data is not required by the user then, it is checked that the user should be able to remove that data manually from the plot list by deleting the corresponding image from the list.
- **Automatic** The data is generated from the plot using automatic mode which uses the internal axis detection and legend detection modules without any user input. It is checked that that the final tables are being generated properly using the internal modules to get the plot information.
- **Manual** If the user is not satisfied with the output generated through automatic mode then , the user can manually input information such as scale for the graphs, number of plot lines, select area of plot to extract plots from the pdf, etc. such that the data extracted from the graph is based on the information entered by the user. This is mainly used in the case of low quality images and/or when OCR engine is not able to obtain the required text.
- **Save** The extracted data from the input pdf is saved in a PDF document with one plot per page followed by the corresponding data tables.
- The interfaces to ensure they are functioning as desired (i.e. check if each interface is behaving as expected, specifically verifying the appropriate action is associated with each mouse_click event).

Performance Testing

This test was conducted to evaluate the fulfillment of a system with specified performance requirements. It was done using black-box testing method. The following were tested:

- Setting high number of graphs in the given pdf file and checking the time required for the extraction of all the graphs and tables for each plot.
- Setting more number of different plot projections per graph and checking the time required for complete extraction and the accuracy of the tables generated for each plot.

- Giving a low quality input pdf and checking the accuracy for the generated tables by automatic mode and then comparing it with the tables generated when some inputs are given by the user manually.

6.6 Entry and Exit Criteria

This section describes the general criteria by which testing commences, temporarily stopped, resumed and completed within each testing phase. Different features/components may have slight variation of their criteria, in which case, those should be mentioned in the feature test plan. The testing phase also maps to the impact level definition when a defect is entered in the bug-tracking phase.

6.6.1 Unit Testing

Unit Testing is done at the source or code level for language-specific programming errors such as bad syntax, logic errors, or to test particular functions or code modules. The unit test cases shall be designed to test the validity of the programs correctness.

Black Box Phase

Black box testing typically involves running through every possible input to verify that it results in the right outputs using the software as an end-user would. We will use Equivalence Partitioning and Boundary Value Analysis complexity metrics in order to quantifiably determine how many test cases needed to achieve maximum code coverage.

Black Box Entry Criteria

The Black Box Entry Criteria will rely on the component specification, and user interface requirements. Things that must be done on entry to the Black Box stage:

- All functions like select PDF, run, deleting extracted plot, automatic, manual, save as PDF functions must be either coded or stubs written.
- The type of Black Box testing Methods will be determined upon entry. We will use Error Guessing, and Boundary Value Analysis.
- Error Guessing included giving low quality pdf as input or pdf containing discontinuous graphs.
- Bounded Value Analysis included giving a pdf with high number of graphs and/or with each graph containing high number of different colored plot projections as input for maximum number of tables to be displayed.

Black Box Exit Criteria

The Black Box Exit Criteria listed below explains what needs to be completed in-order to exit Black Box phase. To exit the Black Box phase 100% success rate must be achieved. Things that must be done upon exiting the Black Box stage:

- The application does not generate any table if no graphs are found in the input PDF.
- The application was tested for large input PDF beyond which a crash might occur which was handled.
- All code bugs that are exposed are corrected.

White Box Testing

The White Box criteria apply for purposes of focusing on internal program structure, and discover all internal program errors. Defects will be categorized and the quality of the product will be assessed.

White Box Entry Criteria

The White Box Entry Criteria will rely verifying that the major features work alone but not necessarily in combination; exception handling will not be implemented. The design and human interface are stable. Things that must be done on entry to the White Box stage:

- All functions like select PDF, run, deleting extracted plot, automatic, manual, save as PDF must be coded.
- The type of White Box testing Methods was determined upon entry. We will use unit testing and test for memory leaks.
- Black Box Testing should be in its late stages.

White Box Exit Criteria

The asfd in the White Box stage should have a generally stable feel to it. White Box testing continued until the Black Box or next milestone criteria were met. To exit the White Box phase 100% success rate must be achieved. The following describes the state of the product upon exit from the White Box Stage:

- All functions like select PDF, run, deleting extracted plot, automatic, manual, save as PDF are implemented and tested
- All Branch Testing test cases were generated. The test cases will be generated from the Control Flow diagrams of all functions.

- The graphical interface has been reviewed and found to satisfactory and is stable, that is, no further changes to any dialog boxes or other interface elements are planned.
- All code bugs that are exposed are corrected.

6.6.2 Integration Testing

There are basically two components of modules that will be integrated for Integration testing. They are The Graphic User Interface module and the modules present in back-end code. The two components consists of a mixture of stubs, driver, and full function code. The following describes the entry and exit criteria for Integration testing.

Integration Test Entry Criteria

The Integration Test Entry Criteria will rely on these modules to be operational. The user interface must be stable. Things that must be done on entry to the Integration Test stage:

- All functions like select PDF, run, deleting extracted plot, automatic, manual, save as PDF are coded.
- The Graphical User Interface is coded using PyQt. The GUI is made such that to facilitate test case input and output PDF files.
- Interfaces and interactions between the back-end modules and the Graphical User Interface must be operational.
- A bottom-up Integration Test Strategy will be conducted. The low level details of back-end modules and graphical interface were integrated. A driver will be written to facilitate test case input and output values. The driver temporarily satisfied high-level details of the input and output values.

Integration Test Exit Criteria

The Integration Test Exit Criteria relied on these X modules to be operational.To exit the Integration Testing phase 100% success rate must be achieved. Things that must be done on exit from the Integration Test stage:

- All code bugs that were exposed were corrected.
- All back end modules like PDF to Image, Graph Extraction, Axis Detection, OCR Engine, Scale calculation, Legend detection, Plot line detection, value calculation and Values to PDF, and Graphical User Interface Module interacted together with complete accuracy, according to the System Specication Design. All discrepancies were corrected.
- All modules are ready for testing. Black Box Testing was completed.

6.6.3 System Testing

The System Test criteria apply for purposes of categorizing defects and the assessing the quality level of the product. All elements of the back end modules and Graphical User Interface are meshed together and tested as a whole. System test focuses on functions and performance, reliability, installation, behavior during special conditions, and stress testing.

System Test Entry Criteria

- Unit testing of all the modules has been completed.
- System test cases have been documented.
- User Interface and functionalities to be tested should be frozen.
- Priorities bugs have been fixed.

System Test Exit Criteria

- Application meets all the document requirements and functionalities.
- Defects found during System testing should be fixed and closed.
- All the test cases for the system have been executed and passed.
- No critical defects have been opened which is found during system testing.

6.7 Sample Test Cases

The following table shows the values obtained for a graph before testing and values obtained for the same graph after testing.

Before Testing		After Testing	
x	y	x	y
-40.6556	-54.2	-40.6556	-54.2
-38.5756	-9.6	-38.5756	-9.55
-37.5356	-4.7	-37.5356	-4.85
-37.0156	-3.05	-37.0156	-3.3
-34.9356	1.05	-34.9356	1.05
-34.4156	1.9	-34.4156	1.95
-31.2956	5.8	-31.2956	5.8
-30.7756	5.95	-30.7756	6.0
-27.6556	8.0	-27.6556	8.0
-21.9356	11.0	-21.9356	11.0

6.8 Deliverables

Following is a list of deliverables for our software: asfd

- Source code
- Installer program to install software
- Software Requirements Specifications
- User Manual
- Installation Guide
- Description of Algorithms
- Modules and Software Architecture
- Test plan document - this document should address testing objectives, criteria, standards, schedule and assignments, and testing tools.
  - Unit Testing Plan
  - Integration Plan
  - System Testing Plan

References

[1] http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0005738

[2] "Scandal Rocks Scientific Community". Deutsche Welle. 30 September 2002.

[3] Blostein, Dorothea, Edward Lank, and Richard Zanibbi. "Treatment of diagrams in document image analysis." *Theory and Application of Diagrams.* Springer Berlin Heidelberg, 2000. 330-344.

[4] Terrades, Oriol Ramos, and Ernest Valveny. "Radon transform for linear symbol representation." *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on.* IEEE, 2003.

[5] Henderson, Tom, and Lavanya Swaminathan. "Symbolic pruning in a structural approach to engineering drawing analysis." *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on.* IEEE, 2003.

[6] Lank, Edward H. "A retargetable framework for interactive diagram recognition." *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on.* IEEE, 2003.

[7] Lu, Xiaonan, et al. "Automatic categorization of figures in scientific documents." *Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries.* ACM, 2006.

[8] Yan, Luo, and Liu Wenyin. "Engineering drawings recognition using a case-based approach." *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on.* IEEE, 2003.

[9] Zhou, Yan Ping, and Chew Lim Tan. "Hough technique for bar charts detection and recognition in document images." *Image Processing, 2000. Proceedings. 2000 International Conference on.* Vol. 2. IEEE, 2000.

[10] https://github.com/tesseract-ocr/tesseract