

---

# OpenSoft Report

---

## TABulator

Submitted by:  
Group 2



Indian Institute of Technology, Kharagpur

May 5, 2016

# Contents

<b>1</b>	<b>Installation Guide</b>	<b>2</b>
1.1	System Requirements . . . . .	2
1.2	Set up Configuration and Setup . . . . .	2
1.3	Packages installed alongside . . . . .	2
1.4	Troubleshooting . . . . .	2
1.4.1	Command <i>tabulator</i> not working . . . . .	2
1.4.2	Dependencies not installed . . . . .	2
1.4.3	OpenCV installation failure . . . . .	3
1.4.4	Version conlict of dependencies . . . . .	3
<b>2</b>	<b>User Manual</b>	<b>3</b>
2.1	Interface of TABulator . . . . .	3
<b>3</b>	<b>Description of Algorithms used</b>	<b>4</b>
3.1	General logic flow . . . . .	4
3.2	Plot rectangle finding algorithm . . . . .	5
3.3	Finding X markings algorithm . . . . .	6
3.4	Finding X label algorithm . . . . .	6
3.5	Finding Y markings algorithm . . . . .	7
3.6	Finding Y label algorithm . . . . .	7
3.7	Finding plot name algorithm . . . . .	8
3.8	Legend detection algorithm . . . . .	8
3.9	Data extraction algorithm . . . . .	9
<b>4</b>	<b>Architectural diagram and Modules of the code</b>	<b>9</b>
4.1	Architecture diagram . . . . .	9
4.2	Sequence diagram . . . . .	10
4.3	Activity diagram . . . . .	11
<b>5</b>	<b>Test Plan</b>	<b>11</b>
5.1	Introduction . . . . .	11
5.1.1	Description . . . . .	11
5.1.2	Hardware Requirements for testing . . . . .	12
5.1.3	Software Requirements for testing . . . . .	12
5.2	White Box Testing . . . . .	12
5.2.1	PDF to image conversion . . . . .	12
5.2.2	Rectangle detection testing . . . . .	12
5.2.3	Legend Detection . . . . .	12
5.2.4	Axes Detection . . . . .	12
5.2.5	Plot Name Detection . . . . .	12
5.2.6	Data Extraction . . . . .	12
5.2.7	PDF Conversion . . . . .	13
5.3	Black Box Testing . . . . .	13

# 1 Installation Guide

## 1.1 System Requirements

- Platform: Ubuntu 14.04 or higher
- Python 2.7

## 1.2 Set up Configuration and Setup

For installing TABulator, the user needs to take the following steps:

1. Enter the directory Team 2
2. Open terminal and run command `sh installer.sh`

At the end of these steps, the software and its dependencies gets installed on the users machine.

## 1.3 Packages installed alongside

Following packages are installed during the installation process of TABulator.

- Imagemagick
- Ghostscript
- Tesseract 3.1
- Python 2.7
- OpenCV 3.1.0
- ReportLib 3.0

## 1.4 Troubleshooting

### 1.4.1 Command *tabulator* not working

Please go to the directory *tabulator* in the submission folder and execute the command `python tabulator.py --inp = /path/to/input/pdf -- out = /path/to/output/pdf`

### 1.4.2 Dependencies not installed

In case one of the dependencies required for TABulator does not get installed, you may find them listed in the previous sections and try installing them manually.

### 1.4.3 OpenCV installation failure

The installation for OpenCV may fail due to system issues. In that case, OpenCV 3.1.0 must be manually installed for the application to work.

### 1.4.4 Version conflict of dependencies

Uninstall the existing version of the conflicting dependency and then install the latest version.

## 2 User Manual

### 2.1 Interface of TABulator

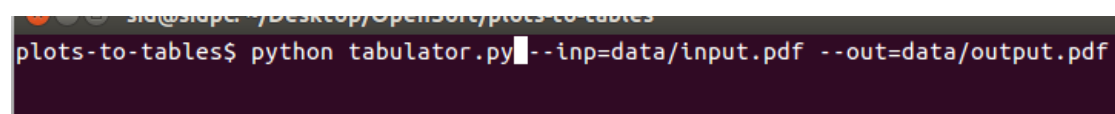
TABulator works on a command-line interface, providing the user with a clean, simple approach to data generation from the documents.

The required processing is carried out by going to the directory **tabulator** and running the command **python tabulator.py**, with a number of command-line parameters to provide the user multiple options for carrying out his desired task.

The command-line parameters available are:

- **inp**: The user can specify the path to one or more input PDFs in this argument. In case of a single PDF, the user simply enters the path to the PDF. In the case of multiple PDFs, the user must add all those PDFs to a folder and write `folderPath/*.PDF` in this argument, where `folderPath` is the path of the folder.
- **out**: Path to the output PDF.
- **csv**: The user can provide the path to a folder in this option, which, if provided, causes TABulator to write CSV data files as well to this folder. This is enabled by default.
- **no\_interrupt**: In the event that OCR fails on given PDF/s, the application by default prompts the user to enter some data, namely, the start of x-axis, value of one division of x-axis, the start of y-axis and value of one division of y-axis. If the user is not interested to enter such data at the moment and only desires to acquire a qualitative analysis of the plots in the document, this option forces the application to make assumptions regarding the missing data and does not prompt the user to enter anything.

A typical command may look like:



```
sid@sidpc: ~/Desktop/OpenSofT/plots-to-tables
plots-to-tables$ python tabulator.py --inp=data/input.pdf --out=data/output.pdf
```

### 3 Description of Algorithms used

We have used the Tesseract OCR engine for detecting the numbers on X and Y axis. However, low accuracy has been observed for Tesseract in some cases. In those cases, we prompt the user to enter some data.

In general, we have observed that Tesseract has low accuracy on text elements of the plots, such as the labels, etc. For solving this issue, we have extracted the labels and plot names directly from the PDFs to add to the output table, as compared to converting them to text and adding them later.

#### 3.1 General logic flow

##### **ALGORITHM**

- 1 Input
- 2 Convert PDF to PNG
- 3 Plot Detection
- 4 For each plot:
  - 5 Find Xaxis()
  - 6 Find Xlabel()
  - 7 Find Xmarkings()
  - 8 Find Yaxis()
  - 9 Find Ylabel()
  - 10 Find Ymarkings()
  - 11 Find WholePlotName()
  - 12 Find legend
  - 13 Find Plot labels
  - 14 Find hue values
  - 15 Extract data from plot
  - 16 Add Page to output PDF
- 17 Write outputPDF

### 3.2 Plot rectangle finding algorithm

#### **Plot Detection**

Input: Image(Page of PDF converted to PNG)

```
1   rectangles=findRectangles(image) //Contour Detection
2   Mask=getColoredMask(image) //coloured pixels
3   Plots=[ ].
4   for rectangle in rectangles
5       if isColored(rectangle)
6           Plots.add(rectangle)
7   return Plots
```

### 3.3 Finding X markings algorithm

#### Finding X markings

Input: image (Page of PDF converted to PNG)

rectangle (bounding box for plot)

- 1 cropped = region in image just below rectangle with same width as rectangle and 10% height as rectangle.
- 2 Dilate cropped only in X direction.
- 3 contours = findContours(cropped)
- 4 numbers = [ ] , thresholdArea=1% of cropped area
- 5 for cnt in contours
- 6     if cnt.area()>thresholdArea
- 7         numbers.add(cnt)
- 8 Remove Noisy values by finding center of X Markings and take those numbers that are near that center
- 9 Find Pixel distance between markings by finding the distance between centers of numbers.
- 10 if OCR engine ran successfully on atleast 2 numbers:
- 11     use them to find xStart and xDelta.
- 12 else : Ask user for input if interrupt is enabled.
- 13 return xStart, xDelta, PixelWidth.

### 3.4 Finding X label algorithm

#### Find Xlabel

Input: image (Page of PDF converted to PNG)

rectangle (bounding box for plot)

- 1 cropped = Region of image below the markings of rectangle.  
Same width as rectangle.10% height as rectangle.
- 2 Dilate cropped in X Direction
- 3 contours = findContours(cropped)
- 4 xLabel = Contour with max area
- 5 Mark the region of xLabel in original image so that any other plot does not use this as plotName
- 6 return xLabel

### 3.5 Finding Y markings algorithm

#### Finding Y markings

Input: image (Page of PDF converted to PNG)

rectangle (bounding box for plot)

- 1 cropped = region in image just left of rectangle with same height as rectangle and 10% width as rectangle.
- 2 Dilate cropped only in Y direction.
- 3 contours = findContours(cropped)
- 4 numbers = [ ] , thresholdArea=1% of cropped
- 5 for cnt in contours
- 6     if cnt.area()>thresholdArea
- 7         numbers.add(cnt)
- 8 Remove Noisy values by finding center of Y Markings and take those numbers that are near that center
- 9 Find Pixeldistance between markings by finding the distance between centers of numbers.
- 10 if OCRengine ran successfully on atleast 2 numbers
- 11     use them to find yStart and yDelta.
- 12 else : Ask user for input if interrupt is enabled.
- 13 return yStart, yDelta, PixelWidth.

### 3.6 Finding Y label algorithm

#### Find Ylabel

Input: image (Page of PDF converted to PNG)

rectangle (bounding box for plot)

- 1 cropped = Region of image left of the markings of rectangle.  
Same width as rectangle.10% width as rectangle.
- 2 Dilate cropped in Y Direction
- 3 contours = findContours(cropped)
- 4 YLabel = Contour with max area
- 5 Mark the region of YLabel in original image so that any other plot does not use this as plotName
- 6 return YLabel



### 3.7 Finding plot name algorithm

#### Finding Plot Name

Input: image (Page of PDF converted to PNG)

rectangle (bounding box for plot)

- 1      cropped = Region of image below the markings of rectangle.  
Same width as rectangle.10% width as rectangle.
- 2      Dilate cropped in X Direction
- 3      contours = findContours(cropped)
- 4      if a large contour is found, that is the plotName
- 5      else: cropped = Region in Image above rectangle
- 6      Find the largest Contour  
That is the plotName
- 7      Mark the region of plotName in image so that any other plot does not use this as  
plotName
- 8      return plotName

### 3.8 Legend detection algorithm

#### Find Legend

Input: image (Page of PDF converted to PNG)

rectangle (bounding box for plot)

- 1      mask = Mark inside original image containing only Block Pixels
- 2      Remove Noise in mask
- 3      Now mask represents the text inside the plot. That is the region which contains the  
plotNames inside the legend.
- 4      Dilate this mask in X Direction
- 5      We get blobs representing individual plotNames.
- 6      if these are coloured horizontal lines to right of the blobs; then those are the colours  
of the plots.
- 7      else: the coloured horizontal line are to the left. Find those colours.
- 8      return plotNames,colours

### 3.9 Data extraction algorithm

#### Extract Data

Input: image (Page of PDF converted to PNG)

hues[] (hue values of legend colours)

jumpInX(interval in X axis at which data points are required)

```
1    plotHSV = convert plot to HSV domain
2    for x in range (0,maxX,jumpInX)
3        for each colour
4            for y in range(0,height)
5                if plotHSV[y][x].hue is near hue of colour
6                    points[colour].add((x,y))
7            if no y found for colour
8                points[colour].add((x,-))
9    return points
```

## 4 Architectural diagram and Modules of the code

### 4.1 Architecture diagram

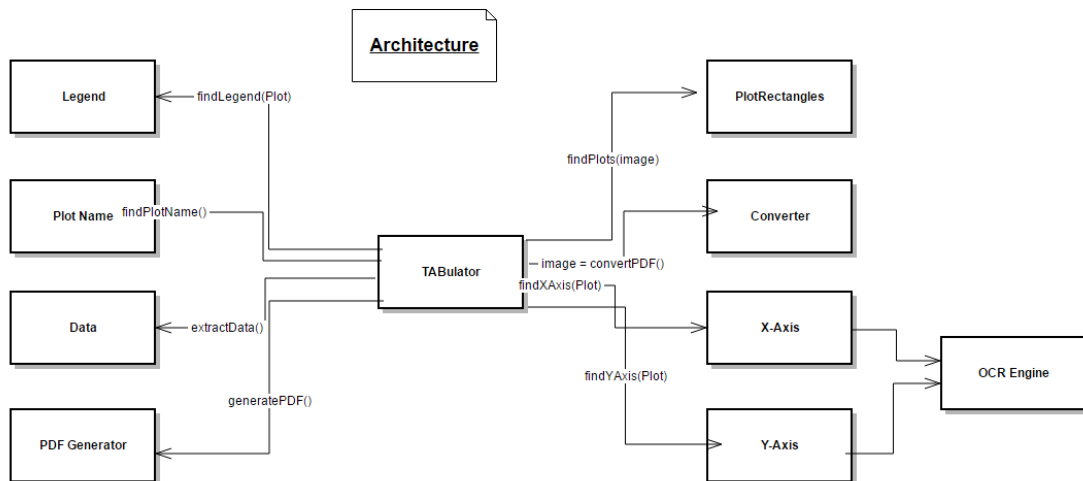


Figure 1:  
Architecture diagram on TABulator

## 4.2 Sequence diagram

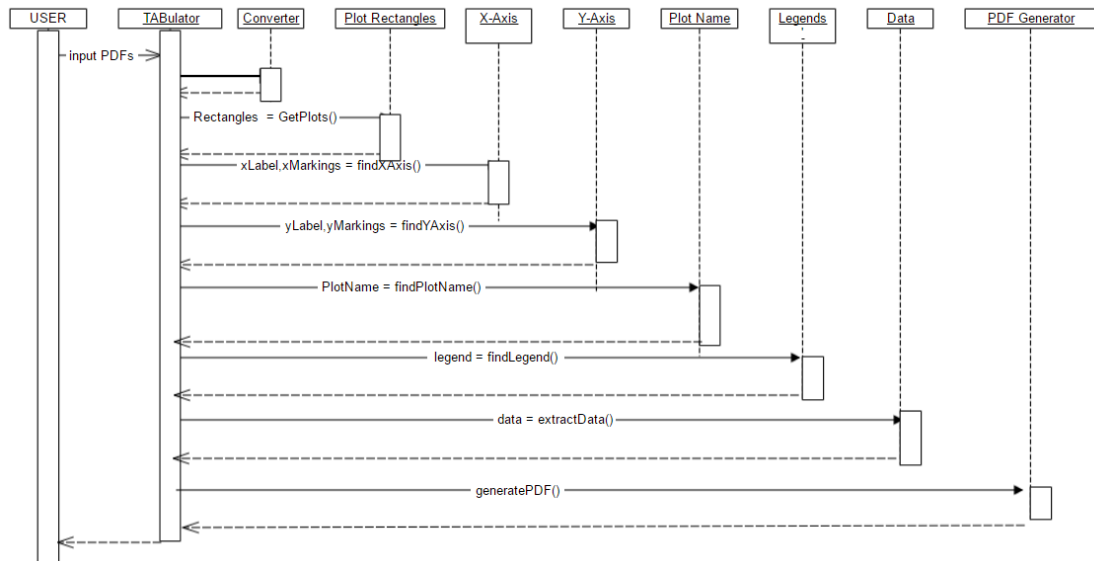


Figure 2:  
Sequence diagram on TABulator

## 4.3 Activity diagram

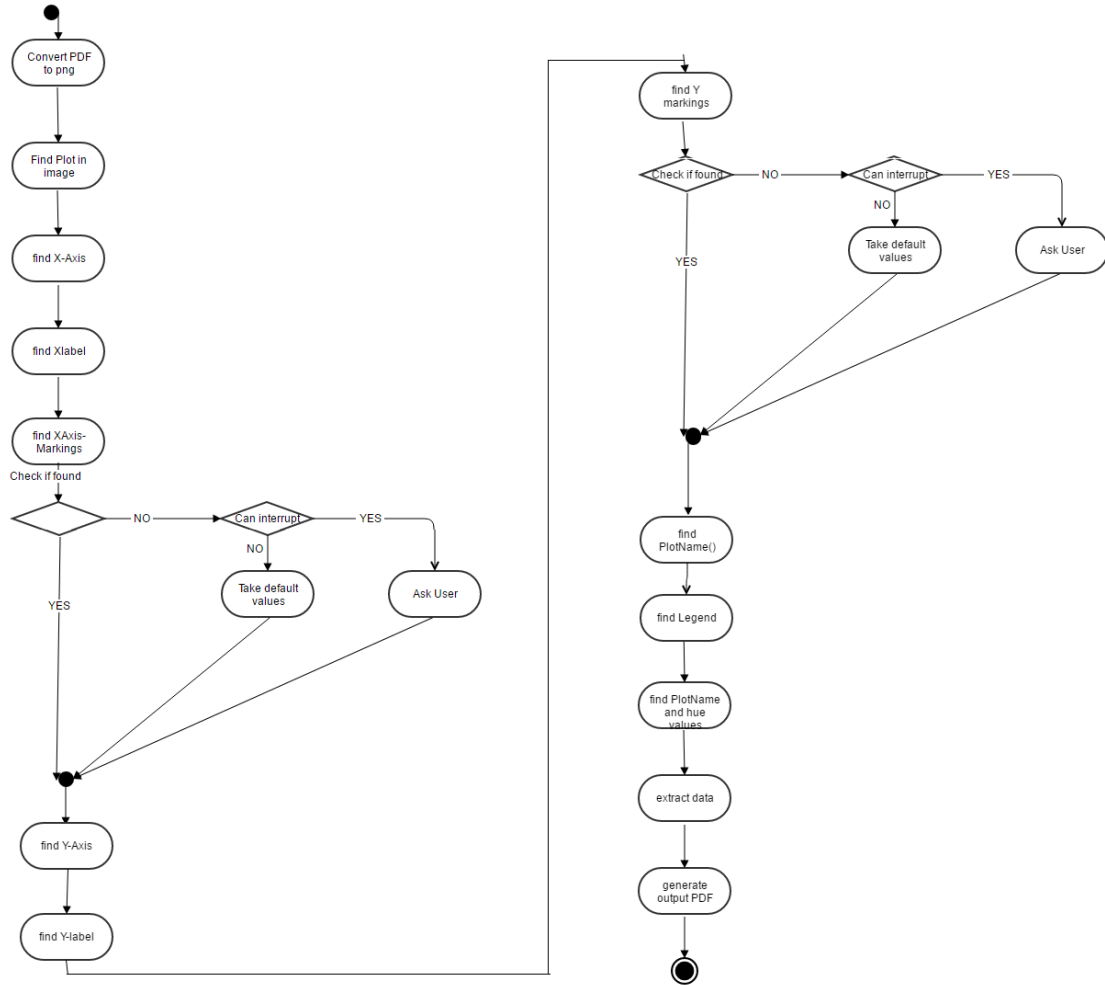


Figure 3:  
Activity diagram on TABulator

## 5 Test Plan

### 5.1 Introduction

#### 5.1.1 Description

This document sketches out the test plan for the TABulator. The test plan used the black box approach and white box approach to uncover various bugs in the software. It was necessary then to perform regression testing to ensure that no new bugs have been introduced due to the introduction of correction patches in the code.

This paradigm will include, but is not limited to, the testing criteria, methods, and test cases of the overall design. Throughout the testing process the test

documentation specifications described in the IEEE Standard 8291998 for Software Test Documentation will be applied.

### **5.1.2 Hardware Requirements for testing**

The software is assumed to be tested on a machine with following conguration:

- RAM: 4GB
- Processor: 64 bit

### **5.1.3 Software Requirements for testing**

Operating System: Ubuntu 14.04 or higher.

## **5.2 White Box Testing**

### **5.2.1 PDF to image conversion**

Different variations of PDFs with versatile graphs were added and tested.

### **5.2.2 Rectangle detection testing**

All sizes of rectangles were detected

### **5.2.3 Legend Detection**

Different variations of legend inputs were tested successfully

### **5.2.4 Axes Detection**

- Different variations of varied scales along x and y axis were input.
- The corresponding scales were successfully detected.

### **5.2.5 Plot Name Detection**

- Plot with different placements of plot name were input.
- The corresponding names were successfully detected.

### **5.2.6 Data Extraction**

- Output with high Precision were obtained for high image quality input plots.
- Output showed significant deviation from actual data with poor image quality input due to Tesseract inefficiency.

### 5.2.7 PDF Conversion

Conversion of Data Table Obtained to PDF was successfully tested.

## 5.3 Black Box Testing

Plot type	Output obtained	Output expected
Plot without Legends and Labels	Headerless Output Table	Output Table with no label
Graph with multiple continuous plots	Table with corresponding outputs	Output Table with corresponding values
Graph with single DASHED plot	Correct Output With Extrapolation	Correct Output
Graph with multiple DASHED plot	Correct Output With Extrapolation with respective legends	Correct Output
Graph with mixed plots both continuous and DASHED	Correct Output With Extrapolation with respective legends	Correct Output.
Graph with Poor Image Quality With Interrupt	Correct Output with respective legends	Correct Output
Graph with Poor Image Quality Without Interrupt	Output With Significant Errors	Correct Output
Single Graph on Single Page	Correct Output	Correct Output
Multiple Graphs on single Page	Correct Output with respective legends	Correct Output
Pdf With Multiple Pages	Correct Output	Correct Output