# Graph Extractor
# Documentation

**Abstract**

Graph Extractor is a graph extracting tool from scanned PDF files which runs on Ubuntu Linux distributions. This document contains the user guide describing how to install and run the software and software documentation describing the algorithms, modules of the code/architectural diagram and test plan.

*Preprint submitted to OpenSoft 2016 judges* *March 16, 2016*

**Contents**

## 1. Installation Instructions

Graph extractor can be installed as an application with **graphical user interface** and as a **console application**.

**Before proceeding please ensure the following**:

- Copy the installation folder to the system and then proceed with the installations. The installer will not work directly from external storage devices.
- The installer installs the dependencies using internet.If you are behind a proxy server, setting the $https_proxy and $https_proxy environment variable is a must. Please set the http and https proxy for the system before proceeding.The installer has been tested and if any error occurs it implies either the internet is not working or the http and https proxies have not been set.
- It is recommended to use Ethernet connection over Wi-Fi connection for installation.

### *1.1. Installing the GUI application*

**Note that installing the required libraries for the GUI application takes a long time (maybe around 1 hour on slower connections).**

#### *1.1.1. Requirements*

- **Operating System**: Ubuntu
- **Architecture Type**: 64-bit (preferably)
- **Dependencies**:
  - build-essential
  - cmake
  - libopencv-dev
  - tesseract-ocr
  - tesseract-ocr-eng
  - tesseract-ocr-equ
  - unpaper
  - imagemagick
  - ghostscript
  - latexmk
  - qt5-default
  - qt5-qmake
  - qtbase5-dev-tools
  - libqt5webengine5-dev

*1.1.2. Installation*

- Open the terminal and set the working directory to the submission folder.

```
:~/Submission$ ls
Console  Console_Installer  Documentation.pdf  GUI  Installer
```

- Change directory to the **Installer** directory.

```
:~/Submission$ cd Installer/
:~/Submission/Installer$ ls
dependencies.sh  graph_extractor_install.sh  install.sh  run.sh
```

- Make the file **graph_extractor_install.sh** executable if not already.

```
:~/Submission/Installer$ chmod +x graph_extractor_install.sh
```

- Run the installer with root privileges and export environment variables as follows:

```
:~/Submission/Installer$ sudo -E ./graph_extractor_install.sh
--- Installing Dependencies
...
...
make[1]: Leaving directory '.../Submission/build/graphextractor'
--- Installation Complete
```

- If you see –**Installation Complete** at the end without any errors, the installation is complete. If not, make sure to handle the errors as suggested by the installer. You will see a new folder named **build** in the parent directory which contains the installed software.

```
:~/Submission/Installer$ cd ../
:~/Submission$ ls
build  Console Console_Installer  Documentation.pdf  GUI
Installer
:~/Submission$ cd build
:~/Submission/build$ ls
graphextractor  Graph_Extractor_run.sh  qpdflib ...
```

**If the GUI application is installed successfully, you can skip the next subsection of installing the console application. The next section i.e. User Manual contains detailed guide to how to run the software.**

*1.2. Installing the console application*

**Note that installing the required libraries for the console application takes around 15 to 30 minutes**

### *1.2.1. Requirements*

- **Operating System**: Ubuntu
- **Architecture Type**: 64-bit (preferably)
- **Dependencies**:
  - build-essential
  - cmake
  - libopencv-dev
  - tesseract-ocr
  - tesseract-ocr-eng
  - tesseract-ocr-equ
  - unpaper
  - imagemagick
  - ghostscript
  - latexmk

### *1.2.2. Installation*

- Open the terminal and set the working directory to the submission folder.

```
:~/Submission$ ls
Console  Console_Installer  Documentation.pdf  GUI  Installer
```

- Change directory to the **Console_Installer** directory.

```
:~/Submission$ cd Console_Installer/
:~/Submission/Console_Installer$ ls
dependencies.sh  graph_extractor_install.sh  install.sh  run.sh
```

- Make the file **graph_extractor_install.sh** executable if not already.

```
:~/Submission/Console_Installer$
chmod +x graph_extractor_install.sh
```

- Run the installer with root privileges and export environment variables as follows:

```
:~/Submission/Console_Installer$
sudo -E ./graph_extractor_install.sh
--- Installing Dependencies
...
...
--- Installation Complete
```

- If you see –**Installation Complete** at the end without any errors, the installation is complete. If not, make sure to handle the errors as suggested by the installer. You will see a new folder named **build_console** in the parent directory which contains the installed software.

```
:~/Submission/Console_Installer$ cd ../
:~/Submission$ ls
build_console  Console Console_Installer  Documentation.pdf
GUI
Installer
:~/Submission$ cd build_console
:~/Submission/build_console$ ls
CMakeCache.txt  cmake_install.cmake  ComputerVision ...
```

## 2. User Manual

### *2.1. Running the GUI application*

To run the GUI application:

- Change the directory to **build** directory created by installation.

```
:~/Submission/build$ ls
graphextractor  Graph_Extractor_run.sh  qpdflib ...
```

- Run the application as follows:

```
:~/Submission/build$ ./Graph_Extractor_run.sh
```

The application will execute.

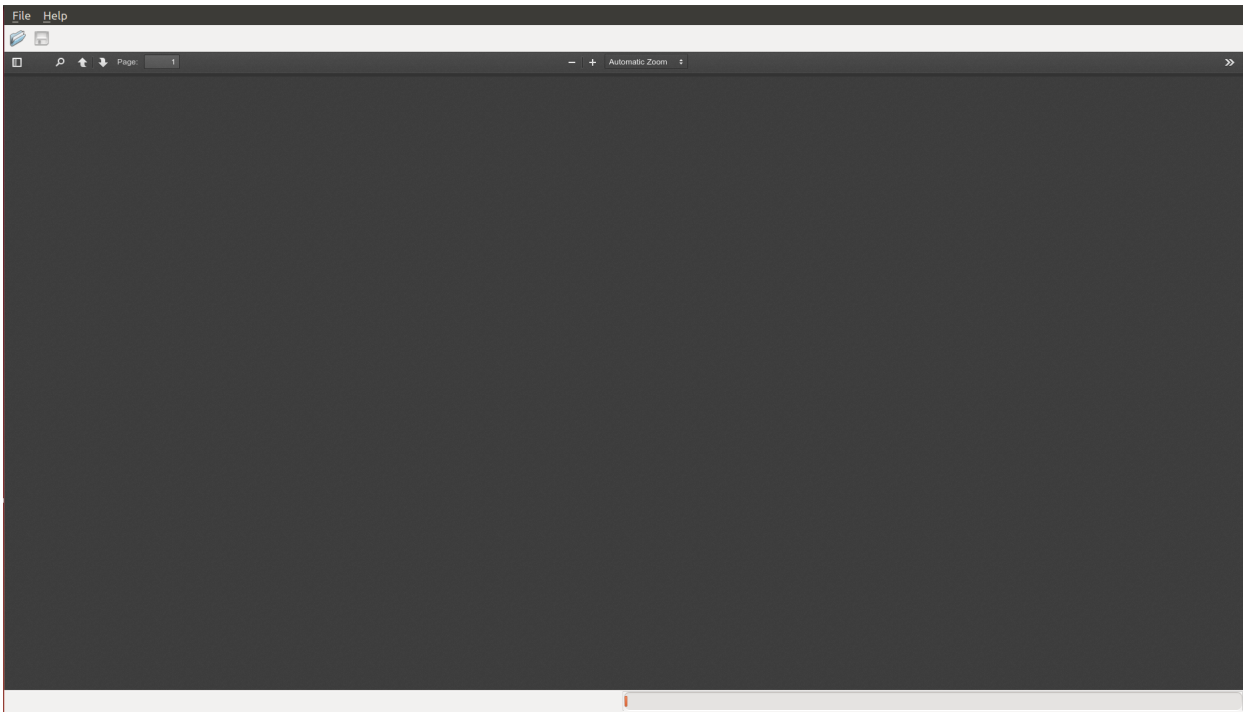


Figure 1: Graph Extractor: Starting the application

*2.1.1. Processing a scanned PDF*

To extract graphs from a scanned PDF and export the tables formed to an output PDF:

- Click the open button or press Ctrl+O or go to File→Open.

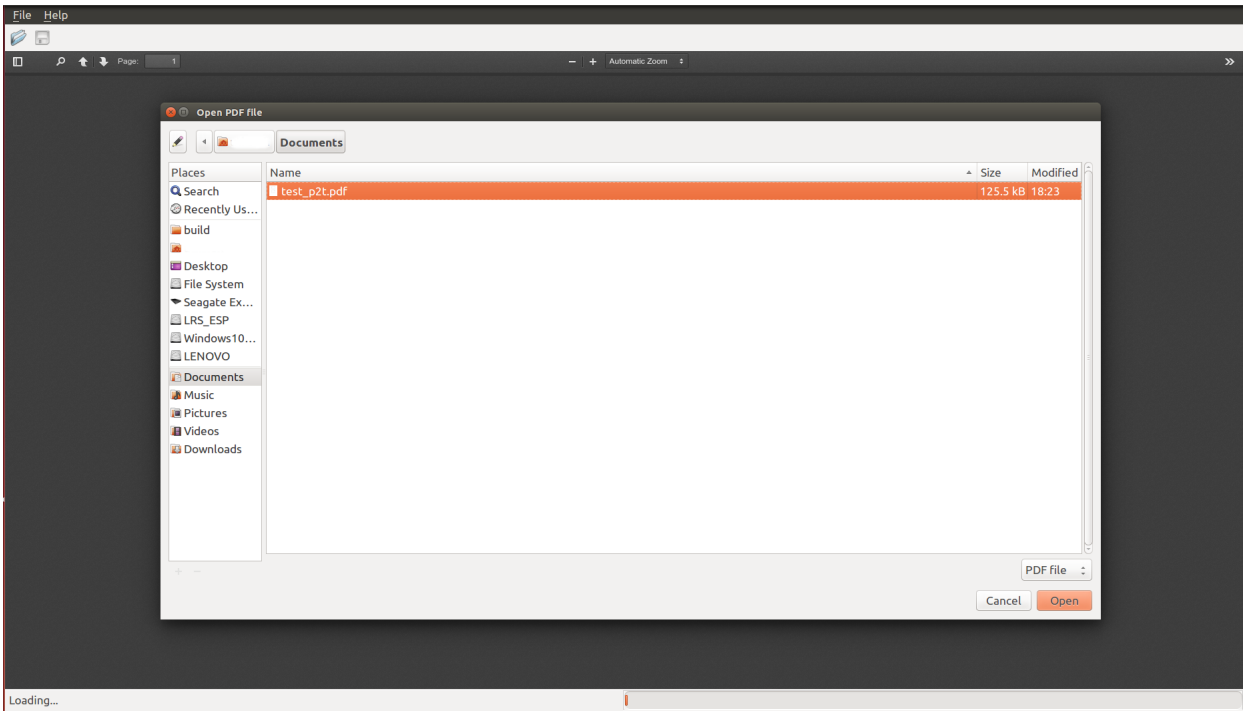


Figure 2: Graph Extractor: Opening a scanned PDF

- Select a PDF file and click **Open** to begin the graph extraction. You will notice taht the status bar at the bottom of the application provides the progress and status of the PDF proccessing.

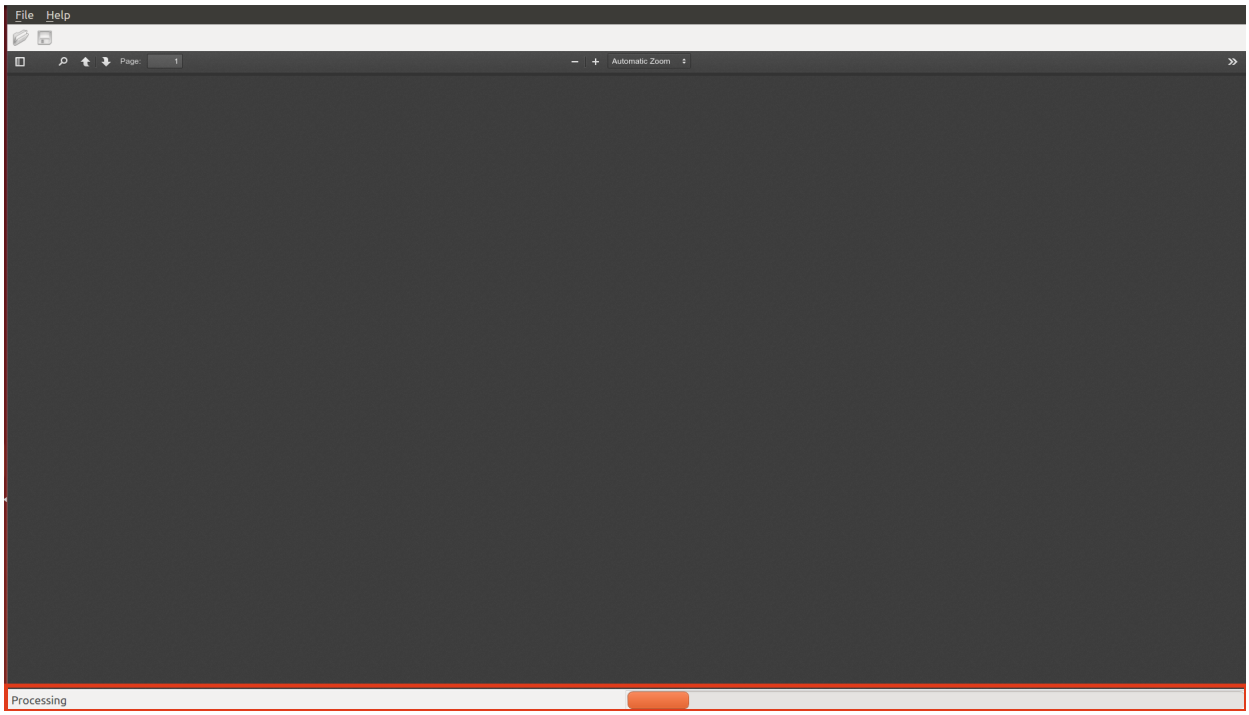


Figure 3: Graph Extractor: Processing a scanned PDF

- After the PDF has been successfully processed, it will be displayed on the viewer.

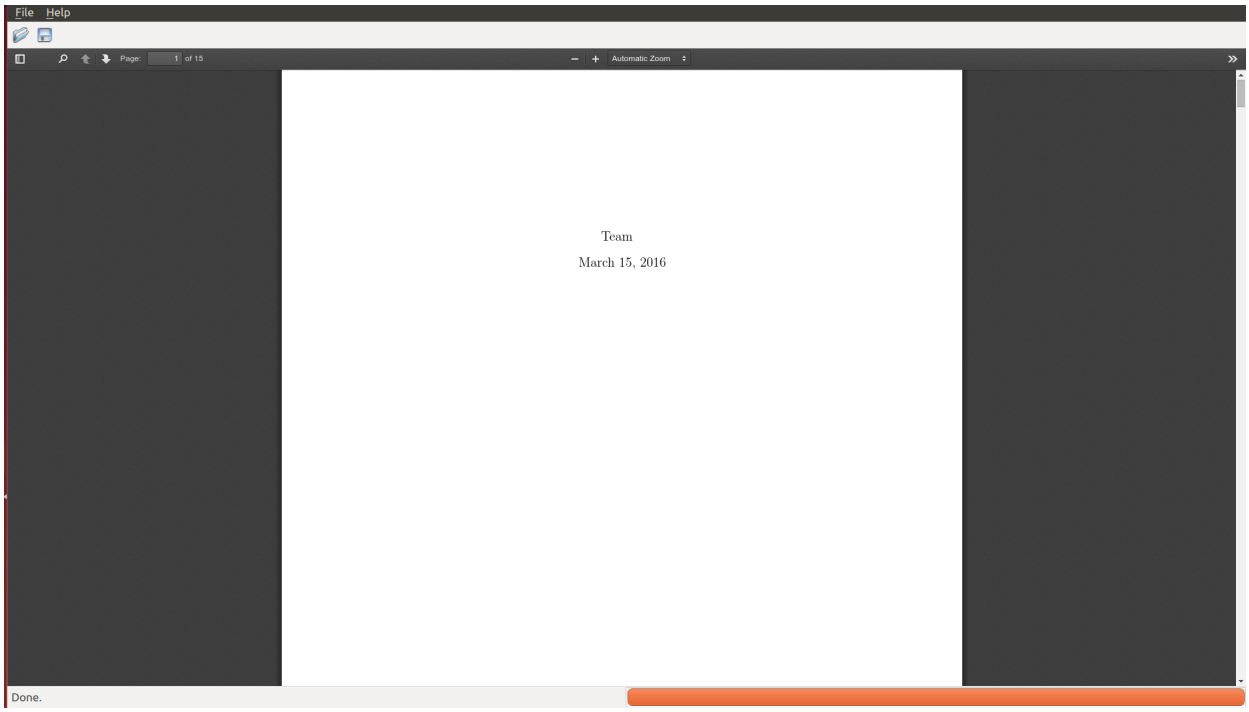


Figure 4: Graph Extractor: Processing a scanned PDF

One can view the complete generated output PDF in the viewer.

*2.2. Running the console application*

To run the console application:

- Open a terminal and change the working directory to **build_console** folder.

```
:~/Submission$ cd build_console/
:~/Submission/build_console$ ls
CMakeCache.txt  cmake_install.cmake  ComputerVision  ...
```

- Run the **run.sh** file with the path of the input pdf as the argument as follows:

```
:~/Submission/build_console$ ./run.sh ~/Desktop/filename.pdf
...
...
100 Done.
```

- The output PDF is generated inside the **build_console** folder inside a folder which is named as *filename*-**dir**(if the filename of input pdf is **scan** then the directory would be **scan-dir**) and named as **out.pdf**. The paths of the output pdf is also the first print output of the program. Open the output PDF with any PDF viewer to view the output.

```
:~/Submission/build_console/filename-dir$ ls
out.aux           out.fls  out.pdf          out.tex     scan-0.png
scan-1-dir  scan-1-table.xml  scan-2.png        scan-3-dir
scan-3-table.xml  scan-4.png out.fdb_latexmk  out.log
out.synctex.gz  scan-0-dir  scan-0-table.xml  scan-1.png
scan-2-dir        scan-2-table.xml  scan-3.png  scan-4-dir
scan-4-table.xml
```

## 3. Algorithm Description

The flowchart of the algorithm of the application is as follows:

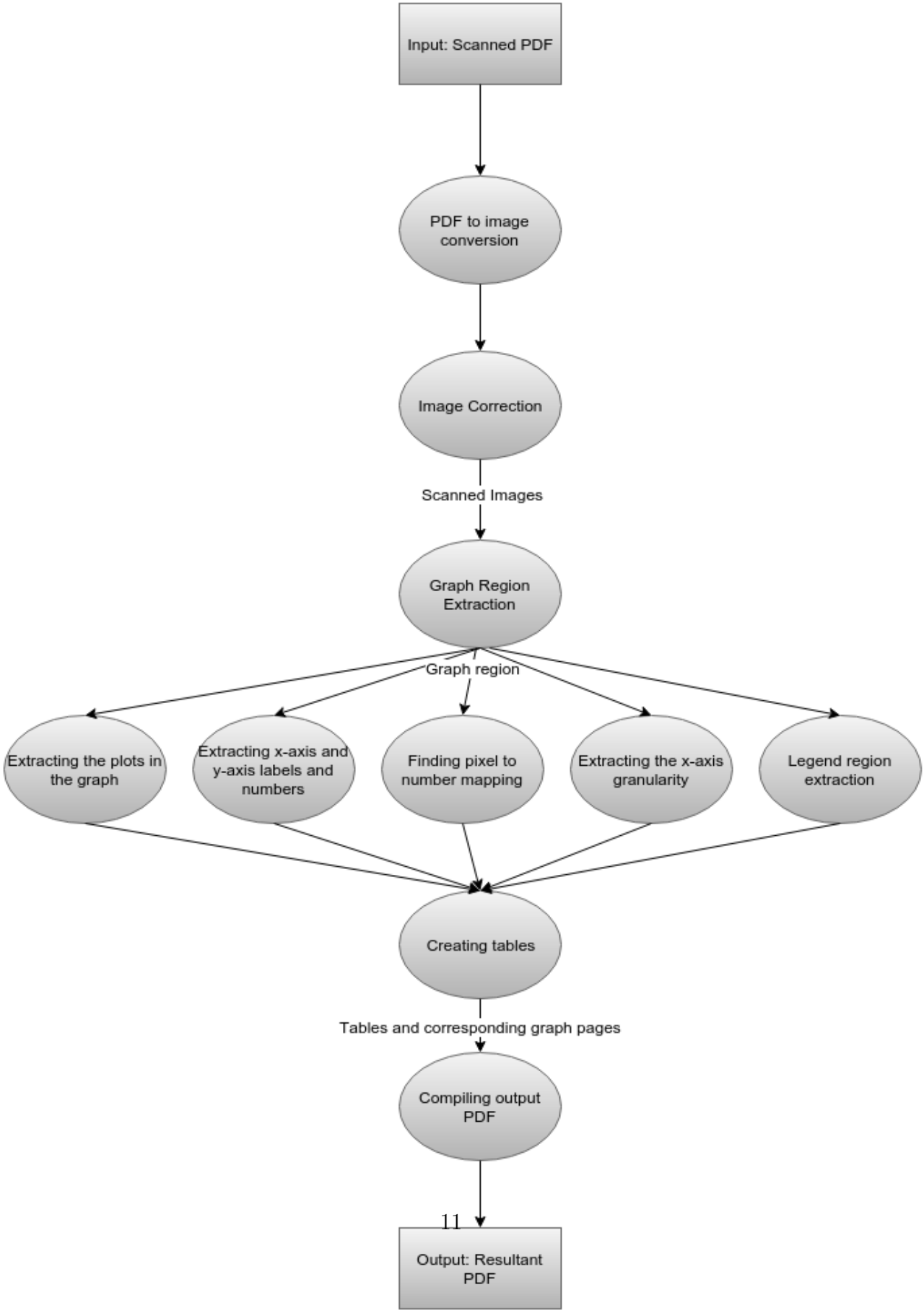


Figure 5: Algorithm Flowchart

*3.1. PDF to image conversion*

First the pages of the PDF are extracted and converted to images using the open-source **ImageMagick**[1] tool which internally uses **Ghostcript**[2] tool. It can be done using a simple *convert* command of the tool. The images are the fed to the image correction module.

*3.2. Image correction*

The images of the page are corrected by this module by applying skew detection using fast fourier transform and skew correction using OpenCV[3]. This is done to improve the output of the OCR engine.The page images after correction are fed to the Graph region extracting module.

*3.3. Graph region extraction*

This module uses a lot of image processing using OpenCV[3]. First it identifies all the rectangular objects in the page. Note that the rectangular objects can be images, tables and graphs.Then it checks for vertical text to the left of the rectangles. If the vertical text is detected, it is certain that the rectangle to the right of the vertical text contains a graph. For the identification of rectangles the module uses OpenCV[3] hough lines to detect all the line segments in the page and the OpenCV[3] contours to approximate a closed figure over these line segments. The closed figures with angles close to 90 degrees are extracted as rectangles. For detecting the locations of the vertical text in the graph we use an open-source tool **Tesseract OCR Engine**[4]. The module feeds the extracted graph regions to further modules for processing.

*3.4. Extracting the plots in the graph*

Different plots in the graph have different colour and in HSV domain(Hue Saturation Value), hue is a good criteria to separate colours as peaks occur at hue values when pixels are coloured. Noise removal is also done to remove anti-aliased pixels. The module uses these peaks to identify the pixels corresponding to each plot in the graph. The module uses OpenCV to do this. It uses spline interpolation as well to recover the missing values. The pixels corresponding to the plots are then fed to the table creating module.

*3.5. Extracting x-axis and y-axis labels and numbers*

This module first identifes where the x-axis and y-axis labels and numbers are. It accomplishes this by first thresholding the graph region in binary (black and white pixels).Then it starts from the top left of the graph region and moves to the right to identify the first vertical region containing the black pixels.This region corresponds to the y-axis label. After identifying the first segment it further moves to the right to identify the next segment containing the black pixels. This region corresponds to the y-axis numbers. The module does a similar process to identify regions of x-axis labels and numbers. These extracted segments are then fed to the open-source Tesseract OCR Engine [4] which gives us the text of the x-axis and y-axis labels and numbers which are then fed to the table creating module and to pixel to graph axis scale mapping module.

*3.6. Finding pixel to graph axis scale mapping*

The task at hand needs the pixel to graph axis scale mapping i.e value/ pixel distance which essentially means it needs to convert the pixel scale to graph axis scale and vice versa to identify a particular 2D coordinate of graph in the graph image. This module handles this task. The module has the x-axis numbers text and their corresponding pixel coordinate from previous module.But the OCR detection module cannot be completely trusted as the text detected might not be correct. Hence, the module takes every pair of numbers detected and calculates the pixel to graph scale and then uses **Ransac algorithm** to remove the outliers. The output scale is fed to the table creating module.

*3.7. Extracting the x-axis granularity*

Now that the modules have the pixel to graph scale, they need the number of pixels between two values on x-axis. To do this, the module first identifies the markings of the x-axis by building histogram and applying threshold at every pixel of x-axis. The markings tend to have higher number of black pixels which can be identified in the histogram. Next, it finds the number of pixels between every consecutive pairs of values on x-axis and feeds the median of these values to the table creating module.

*3.8. Creating tables*

The inputs from the previous modules are used for finding the y-coordinate corresponding to the x-coordinates for each plot in the graph. The tables are built corresponding to each graph and feeded to PDF building module along with their corresponding images.

*3.9. Creating output PDF*

The module formats the input images and tables into LaTeXformat and creates a tex file. It uses open-source tool latexmk [5] to build the output pdf using the tex file generated.

## 4. Software Architecture

## 5. Test Plan

*5.1. Description*

This section describes the testing strategy and approach used to validate the quality of this application.

*5.2. Testing strategy*

*5.2.1. Black box testing*

: The application is tested with black box testing strategy where the application is tested through all the layers of code.

*5.3. Module Testing*

For the intermediate modules ,we render the intermediate images using OpenCV and check the results manually for each page of scanned PDF. The result is considered to be passed if the OCR text/ bounding box are correct in the rendered image.

- **Test**: Test whether output PDF is being generated
- **Strategy**: The output PDF is displayed on the UI after the process finishes and in case of console application it is found to be in the directory where the output directory was specified in the console application.
- **Status**: Passed for all the input PDFs.

---

- **Test**: Test whether output PDF contains tables
- **Strategy**: The output PDF should contain a table after each page containing a graph which was tested on several input PDFs.
- **Status**: Passed for all the input PDFs.

---

- **Test**: Test whether output PDF contains a table corresponding to each graph
- **Strategy**: The output pdf should contain equal number of tables in the next pages and graphs in the scanned page which was verified on several input PDFs.
- **Status**: Passed on good quality scanned PDFs. On PDFs containing low quality images this test may fail.

---

- **Test**: Test whether a table in the output has correct number of columns corresponding to each plot
- **Strategy**: The number of columns in a table corresponding to a graph should be equal to the number of coloured plots in the graph which was verified for various input pdfs.
- **Status**: Passed on good quality scanned PDFs. On PDFs containing low quality images this test may fail.

---

- **Test**: Test whether a table in the output has correct x-y values for each plot
- **Strategy**: This is tested using plotting tools where we upload the data file which is created by copying the table and formatting it as csv file. The plots should look similar to their scanned image counterparts which was verified for several input PDFs.
- **Status**: Passed on good quality scanned PDFs. On PDFs containing low quality images this test may fail.

---

## 6. Known Issues and future improvements

- For PDF containing very low quality images in which text is not clear or graph lines are not clear the application may not detect them and will not create tables correctly in the PDF. The title and axis label text in the tables may also be wrong as the OCR engine used might not detect them correctly. You might see nan values in the table which is because of this. For this one needs to use good quality PDFs where the text and the graphs are much clear and visible.

- For large files (file size around 100 MB) the GUI application always generates output PDF but may fail to render it on the GUI displaying a white screen. This is because the renderer currently loads the complete PDF data in the RAM and the web engine used cannot handle large amounts of data. The solution to this is to load the pages as the user requests it on the screen and not the complete PDF which is to be done in the future. The workaround is to open the PDF manually in folder $build/filename-dir/out.pdf$.

## References


[1] I. Studio, ImageMagick: Convert, edit or compose Bitmap images.
URL `http://www.imagemagick.org/`

[2] A. Software, Ghostscript: an interpreter for the PostScript language and for PDF.
URL `http://www.ghostscript.com/`

[3] Itseez, Opencv.
URL `http://opencv.org/`

[4] Google, Tesseract.
URL `https://github.com/tesseract-ocr`

[5] latexmk.
URL `https://www.ctan.org/pkg/latexmk/?lang=en`