# Read-Pdf

## For processing of the plans, all text in the plan is saved beforehand, and its position is recorded in a database. This allows us to search for and locate fire alarms by their Id. A sharpening filter is then applied to the bitmap afterward.

## Loading Pdf

### The script supports two different file modes for reading. The mode to be used is specified through command-line arguments. In "single" file mode, multiple pages are read from a single PDF, and which pages to read is also set through command-line arguments. In "multi" file mode, multiple PDF files are read from a folder, and command-line arguments determine which PDFs should be read.

### Poppler

### The Python library "Poppler" is used to convert PDFs to bitmap format. Poppler was chosen because it best meets our need for very high reading speed.

## Bitmap Processing

### CV2

CV2 is an open-source computer vision module for Python. It is used for image processing, transformation, and filtering. In the "ConversionLoop()" function, CV2 is used to prepare the bitmaps for the AI. The bitmap is converted into a binary format using the "threshold" function to significantly speed up processing by Tesseract.

Later CV2 is also used in the post-processing step for the sharpening of the Bitmap and noise filtering.

CV2 is also used to handle different cases with rotated files. All files need to be rotated correctly to allow the AI to find Texts better and the deletion oft the Texts from the image only works on Texts written from left to right.

Tesseract

For the Text Recognition in the pre-processed Bitmaps the python library Tesseract is used. Tesseract allows us to extract all Texts in the Bitmap in a single call of the function “image\_to\_data”. We sort out the data by looking at the confidence value outputted from the AI. If it exceeds a threshold set by us it is written in a Database. The recognised Texts are then removed from the Bitmap with CV2.

AI and Neural Networks, specifically LSTM (Long Short-Term Memory) networks, are used to improve text recognition. The LSTM network learns context and patterns in the text to improve accuracy, making it better at detecting even distorted texts.

Firebird

To store all detected texts, we use Firebird as an embedded Database. A Database provides more efficiency because of the large amount of data needed to be looked at to find the right fire alarm. A Database also provides Full-Text-Search capabilities which allows to query the data more efficient without searching for an exact match.

Firebird was chosen because a database was needed which could support a high level of concurrence but still staying lightweight.

To use Firebird as just an embedded Database there needs to be a reference to Firebird in the System-Path variables before “System32”. In the Python code the fdb api must be loaded with the path to “fbclient.dll”. After tis the database can be used just like a normal connected DB.

Because of the missing functionality of the “IF NOT EXISTS” statement there is a workaround used when dropping existing tables. By looking at the “rdb$relations” table and filtering for the “table\_name” it can be figured out if a table already exists to safely drop it without risking unwanted errors.

After all Texts are detected, they are written into The Database. The text is stored as a string and the relative distances from the top and the left side are stored respectively as “cordTop” and “cordLeft”. These coordinates will be used later to search for fire alarms and get their location on the map.

### Sharpening and Noise Reduction

#### Gaussian Filter

To Sharpen the image there is an algorithm inspired by the canny edge detection used.

It starts with a gaussian blur filter. The gaussian filter smooths out the image in a specified radius by taking a weighted average of the surrounding pixels using gaussian normal distribution. The gaussian filter can be manipulated by specifying the radius of the pixels taken into consideration and a value σ which specifies the standard deviation of the normal distribution.

The Gaussian Filter is implemented by running multiple kernels simultaneously with pre calculated weight matrices to improve performance.

#### Thresholding

The sharpening happens in the second Step, a threshold is applied to cut of all values under a specific value to get sharp edges again. Noise which is too small to get picked up again by the threshold after blurring vanishes through this process.

As a result, we get a bitmap with a minimal amount of noise left and all texts removed. This bitmap is then saved for the Helios-Interface to use.

## Configuration

For easy debugging and adaptability no paths are Hardcoded in the Code but taken from a global conf.ini file which is also used by other programs in the Helios Intelligence Module. All folders except the folder which contains the Pdf files are also created dynamically if not existent. The root-Directory is dynamically determined to always give the correct path even when the program is called as a executable from another directory.

Some paths to libraries like firebird, poppler or tesseract are also defined in this file and need to be changed to the corresponding paths on another system.

## Logging

A Logging-Handler is used for the logging. The log level and the used logfile is determined through the conf.ini file. Currently supported log levels are “INFO” and “DEBUG”. The logger writes the logging messages to the console and writes into the log file. The log file has a rotating handler which limits the file size and starts deleting the oldest entries to make place for the new ones once the file reached a size of 5mb. All logging Messages are written with the current timestamp and the logging level for easier debugging.

## Pyinstaller

To make the Program available even on all machines even without a python installation the script is converted int an executable file using the library ”pyinstaller”. All Dependencies are located the folder “\_internal” which must always remain in the same directory as the ReadPdf executable. To start the compiling there is a file in the \build directory of the project called BuildString.txt containing the command which is used to start pyinstaller correctly.

## Usage

This tool is designed to be used from the command line. It utilizes Python’s argparse library to parse arguments. Each argument is identified by a tag, which can be written in either its full form (e.g., --UsedDir) or a shortened form (e.g., -uD). The following syntax should be followed when specifying tags:

|  |  |
| --- | --- |
| --FileMode or -fM | Specifies which File-Mode should be used, options are “single” for a single multipage Pdf, or “multi” for multiple single page Pdf files. |
| --FilePath of -fP | Used in File-Mode single: The path to the PDF file starting from the specified Pdf-Storage Directory, if it is in there without a folder the name of the file is enough. |
| --Rotation or -r | Define how rotation should be handled,  nr -> no Rotation  rr -> Rotate Right (clockwise)  rl -> Rotate Left (counterclockwise) |
| --UsedPages or -uP | Used in File-Mode single: After this tag there should be all the page numbers that should be converted, they do not have to be sorted. |
| --UsedDir or -uD | Used in File-Mode multi: Specifies in which Folder Starting from the Pdf-Storage Directory the different pdf files are stored. |
| --UsedPdf or -uP | Used in File-Mode multi: Specifies the names of the different Pdf files which should be Converted, these files have to be in the directory set by --usedDir |