GOCR-documentation

Jörg Schulenburg[11]

Magdeburg,

Abstract

In this documentation I describe some ideas for my OCR-program. It contains algorithms and examples and gives you an impression of what the program can (or could) do.

1 Joerg.Schulenburg@physik.uni-magdeburg.de

1 Introduction

First I have to say that I am not a expert in pattern recognition or similar sings. My knowledge is based mostly on experiments with my program. Therefore do not worry about stupid algorithms I put in this document. In this documentation I describe some ideas for my OCR-program. The examples give you an impression of how does the program handles your images. If you have comments regarding contents or spelling please write to the author.

2 List of ideas

2.1 Segmentation of textual regions / Layout analysis

This is implemented as a recursiv division in two parts.

- look for the thickest horizontal or vertikal gap through the box
- is the gap less than five times longer than thick do not divide
- do the same with the two new parts

I know that this algorithm is not as good as you wish, but I do not know a better one.
It would be very helpfull to know about a funtion which is able to decide whether the box represents a single text line or a more complex object.

2.2 Line detection

Lines of characters are detected by looking for interline spaces. these are characterized by a large number of non-black pixels in a row. Image rotation presents a problem, therefore the program first looks only at the left half of the image. When a line is found, the left half of the right side is scanned, because lines are often short. The variation in height gives an indication of the rotation angle. Using this angle, a second run detects lines more accurately. Line detection may fail if there is dust on the image.
In version v0.2.3 this behaviour is slightly changed. To detect the rotation angle, the line through the most characters is detected.

2.3 Cluster detection

A cluster is a group of pixels which are connected with each other. The simplest way to detect a cluster is to look for a pixel. If you find one, look to the neighbouring pixels. This can be done recursively.
This method needs a lot of stack space if a cluster is very large, and can cause problems with the memory.
Do you remember the algorithm for leaving a maze? Go along the right (or left) wall. This seems to be a good approach for detecting clusters without recursion. The following picture shows a trace of the maze algorithm.

```
first 35 steps       next 36 steps
..@@@@@..@@@@<..     ..v<<<<..v<<<@..     * = starting point
```

```
..@@@@@@@@@.@^<.    ..>>v@^<<<@.@@@.    >^<v = go
right,up,left,down
....@@@@@...@@^.    ....v@@@@...@@@.    @ = black pixel
....@@@@....@@^.    ....v@@@....@@@.
....@@@.....@@^.    ....v@@.....@@@.
....@@@.....@@^.    ....v@@.....@@@.
...@@@@.....@@^.    ...v<@@.....@@@.
...@@@......@@^.    ...v@@......@@@.
...@@@......@@^.    ...v@@......@@@.
...@@@.....@@@^.    ...v@@.....@@@@.
...@@@.....@@>^.    ...v@@.....@@@@.
...@@@.....@@^..    ...v@@.....@@@..
..@@@@.....@@^..    ..v<@@.....@@@..
..@@@@....@@@^..    ..v@@@....@@@@..
*>>>>>>>>>>>>^<<    @@@@@@@@@@@@@@@@
```

The minimum and maximum coordinates can be used to create a box around the cluster. But does this algorithm work with diagonally connected pixels?

2.4 Engine

How does the engine identify a character? For explanation look at the following pattern. The program looks for simple geometric properties.

```
vvvv            vv- white regions
......@@......   <- crossing one line
......@@......
.....@@@@.....
.....@@@@.....
.....@@@@.....
....@..@@@....   <- white hole / crossing two lines
....@..@@@....   <- crossing two lines
....@..@@@....
...@....@@@...
...@....@@@...
...@....@@@...
..@@@@@@@@@@..   <- horizontal line near center
..@......@@@..
..@......@@@..
.@........@@@.   v- increasing width of pattern
.@........@@@.   v
.@........@@@.   v
@@@......@@@@@
     ^^^-- gap
```

In the future it should detect edges, verteces, gaps, angles and so on. With such data the engine could make a cluster analysis. But this is a difficult task, if the scanned image is noisy.

2.5 Remove pixels

The following picture shows a *n* which has additional pixels at the bottom. Therefore it can not be detected as *n*. What can be done?

- classify horizontal (=) and vertical (I) pixels by comparing the distance between the next vertical and next horizontal white pixels (.)
- measure mean thickness of vertical and horizontal clusters
- erase unusually thin horizontal pixels at the bottom line

```
..@@@@@..@@@@@..       ..==III..===II.. dx=16 dy=15
..@@@@@@@@@.@@@.       ..==III====.III. thickness 2 to 3
....@@@@@...@@@.       ....III==...III.
....@@@@....@@@.       ....III=....III.
....@@@.....@@@.       ....III.....III.
....@@@.....@@@.       ....III.....III.
...@@@@.....@@@.       ...IIII.....III.
...@@@......@@@.       ...III......III.
...@@@......@@@.       ...III......III.
...@@@.....@@@@.       ...III.....IIII.
...@@@.....@@@@.       ...III.....IIII.
...@@@.....@@@..       ...III.....III..
..@@@@.....@@@..       ..IIII.....III..
..@@@@....@@@@..       ..IIII....IIII..
@@@@@@@@@@@@@@@@       ================
        ^^^
        this causes the problem
```

The next picture shows blind pixels which are caused by dust on the paper. The upper right dots are not connected with the rest of the character. This can be detected via fill-algorithms. Currently the program assumes that dots near the upper end of a character are “i”-dots or diaereses (umlaut dots).

```
.........................O...      .........................O...
.........................O...      .........................O...
............................      ............................
............................      ............................
.........@@@.......@@@@.....      .........@@@.......@@@@.....
..@@@@..@@@@@@@...@@@@@@@.....      ..@@@@..@@@@@@@...@@@@@@@.....
@@@@@@@@@@@@@@@@.@@@@@@@@@....      @@@@@@@@@@@@@@@@.@@@@@@@@@....
```

```
..@@@@@@....@@@@@@.....@@@@...
..@@@@.......@@@@......@@@@...
..@@@@.......@@@.......@@@@...
..@@@@.......@@@.......@@@@...
..@@@@.......@@@.......@@@@...
..@@@@.......@@@@......@@@@...
..@@@@.......@@@.......@@@@...
..@@@@.......@@@@......@@@@...
..@@@@.......@@@.......@@@@...
..@@@@.......@@@.......@@@@...
..@@@@.......@@@.......@@@@...
..@@@@.......@@@.......@@@@...
..@@@@.......@@@@......@@@@...
..@@@@......@@@@@......@@@@@..
@@@@@@@@..@@@@@@@@@..@@@@@@@@@
```

```
..@@@@@@....@@@@@@.....@@@@...
..@@@@.......@@@@......@@@@...
..@@@@.......@@@.......@@@@...
..@@@@.......@@@.......@@@@...
..@@@@.......@@@.......@@@@...
..@@@@.......@@@@......@@@@...
..@@@@.......@@@.......@@@@...
..@@@@.......@@@@......@@@@...
..@@@@.......@@@.......@@@@...
..@@@@.......@@@.......@@@@...
..@@@@.......@@@.......@@@@...
..@@@@.......@@@.......@@@@...
..@@@@.......@@@@......@@@@...
..@@@@......@@@@@......@@@@@..
@@@@@@@@..@@@@@@@@@..@@@@@@@@@
```

2.6 Add pixels

The following picture shows a *m*. The legs are only laxly connected. How do we handle this?

- if the engine has failed, a filter is switched on and the engine starts over
- the 2x2 filter sets pixels to (*O*) near laxly connected pixels

```
    vv    vv
@@@.@@@..@@@...       @@@.@@@..@@@...
.@@.@@@@.@@@@..<      .@@O@@@@O@@@@..   filter: .@ => O@       @. =>
@O
.@@@..@@@..@@..<      .@@@..@@@..@@..              @. => @.       .@
=> .@
.@@@..@@@..@@@.       .@@@..@@@..@@@.
.@@@..@@@..@@@.       .@@@..@@@..@@@.
.@@@..@@@..@@@.       .@@@..@@@..@@@.
.@@@..@@@..@@@.       .@@@..@@@..@@@.
.@@@..@@@..@@@.       .@@@..@@@..@@@.
.@@@..@@@..@@@.       .@@@..@@@..@@@.
.@@@..@@@..@@@.       .@@@..@@@..@@@.
.@@@..@@@..@@@.       .@@@..@@@..@@@.
.@@@..@@@..@@@.       .@@@..@@@..@@@.
@@@@@.@@@@.@@@@       @@@@@.@@@@.@@@@
```

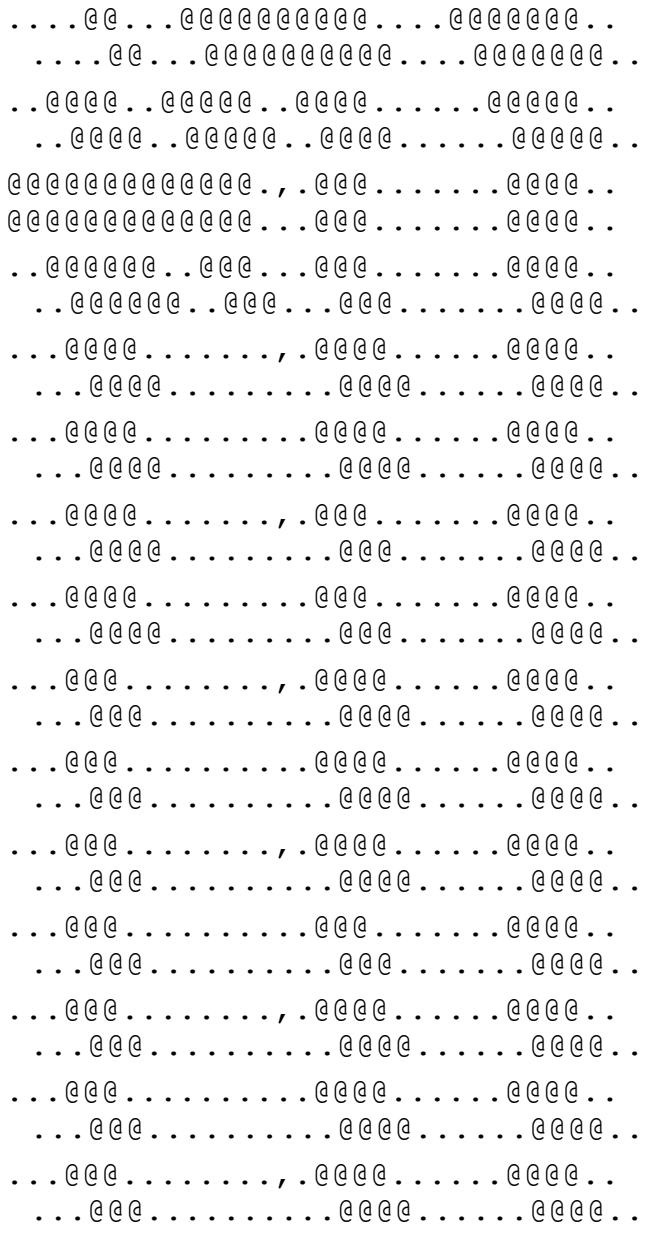
2.7 Similarity analyzer

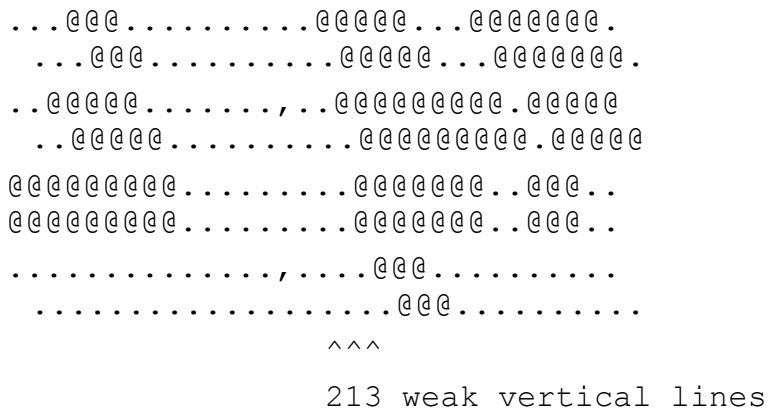
Some characters are a little bit noisy. These characters can be identified by comparison with other, already recognized characters. This can be done via a good distance function.

2.8 Overlapping characters

The following picture shows an overlapping *ru*. How do we handle this?

- look for 3 weak connections (sum over y is small, start in the middle)
- test if the right and left part can be detected by the engine
- correction of surrounding box





Of course the situation is more difficult with slanted characters.

3 Tools

`pbmclean` — This program is written by Angus Duggan and Jef Poskanzer. It cleans up “snow” on bitmap images.