Line Recognition in Text Images

Introduction

Text papers surround us all over: we (well, students) get our bills written on paper; we read newspapers everyday; we sum all sorts of texts while studying for an exam and so on. This is not something special, BUT the innovation of the recent years in the shape of the scanner machine and the cameras in every cell-phone, led to a new area of research: these new devices let us do sophisticating actions on text images if we just know how to process them. It is just natural that we would like first to recognize where the text is placed in an image and how it is ordered. In my project I recognize text lines, as a pre-step to recognize paragraphs. I chose to implement a variation of an algorithm named Docstrum by Gorman. There are several descriptions of the algorithm in the web but not a single implementation, so I thought it could be nice to see how it works.

Approach and Method

The Docstrum algorithm is a bottom-up approach based on nearest-neighborhood clustering of connected components extracted from the document image.

1. pre-processing
2. find atomic connected components and their centers
3. for each component find k-nearest-neighbors and extract in-line neighbors
4. create lines accordingly and improve with some assumptions

Pre-processing

The program may receive any kind of image in respect of color, size and orientation. It changes the image to BW and shrinks it so the program works faster. The main action in this stage is rotating the image. This is done by projecting the pixels on an imaginary axis. When the image is on the right direction there will be clear distinguished areas of black and white. When the image is rotated we'll get a spread projection. We rotate the image on several angles and choose the one that more fits to the distinguished model. I used an implementation of Abed Asi.
Find atomic connected components (characters) and their centers

Connected components and their centers are found with functions from the `regionprops` option in MATLAB. In this stage, very large and very small components are removed. Large components are thought to be pictures which are not of our interest, and small components can be noises or some kind of punctuation signs (', the dot of the 'i' character, punctuation in other languages such as Hebrew and Arabic). All of these are distracters for the find-centers process (that aims to find the center of the lines) that comes afterwards and therefore should be removed.

K-nearest-neighbors and extract in-line neighbors

For each component we find the k-nearest-neighbors. I chose k=12 for my examples. Nearest neighbors are measured by Euclidean distance of the centers. So now each component has a set of his neighbors. In order to extract the in-line neighbors from this set (neighbors that are in the same line as you), angles are calculated between the component
and each of its neighbors (angle between the vector composed of the two centers and the x-axis).

It's hard to represent this in picture... here is an example of all neighbors map, reduction to in-line neighbors and the representation on image. We'll look on the neighbors of component #26. In the picture it's marked in light-blue. In line 26 you can see its KNN.

below are the neighbors of #26 that are in the same line

component #26 and its component neighbors #6, #32, #45 (sorry for the colors)
Create lines

In this stage we perform some kind of union-find algorithm: If two different components have a mutual in-line neighbor, they are both in the same line as well. This is done by a bit complicated recursion, while in each "union step", all in-line neighbors are marked with the same color. There's a small filter in this step that marks as background lines that are composed of one component (they were probably some noise that was not filtered before). This union is not perfectly done. If a real line has some large space within it, it will appear as two separate lines. The next stage suppose to take care of this problem.

Improve lines

This stage aims to fix the problem mentioned before of a real line that was recognized as two or more lines. It checks if two lines that are suspicious to be one have the same line from above and beneath. If so, it unions them. It sometimes union two lines to be one if the lines are close to each other.

All sub-lines are now merged to a single line.
Problems

Distance between components is not taken into account: In the "find KNN" stage components of different columns of the text may be marked as neighbors if they are last and first in the same line; In the "improve lines" stage two lines may be marked as one if two components (one of each line) have a neighbor in a same third line.

These things can be fixed with not much effort if Euclidean distance will be take into account.

Conclusions

This proof of concept works well for very specific type of texts: with large space between lines, large space between columns and a relatively constant space between characters. In order to use this algorithm for more varied texts, the algorithm should be tuned.