

Detect text orientation



How to detect text orientation in an image?

It doesn't matter if the orientation is upside down (180 deg).. But if the text lines is vertical (90 or 270 deg) I need to rotate it 90 degrees.

I hope its possible without OCR because it takes too much resources to process OCR on 4 different orientations of the same image

The reason is that I use scantailor on images from a digital camera or smart phone and if the text orientation is 90 or 270 degree sometimes the image is cropped and text is lost

[image-processing](#) [text](#)

asked May 21 '14 at 12:23



[clarkk](#)

5,581

32

118

187

2 Answers

The proposed solution (Hough transform) is good (and I upvoted it) but it might be CPU intensive. Here is a quick dirty solution:

1. Just calculate a horizontal projection (sum the brightness of the pixels in each pixel row). It should clearly mark the positions of the text lines (bonus: you get a partition of the text to lines). Do otsu binarization to clearly see the partition.
2. Rotate the image by 90 degrees and repeat step 1. If now the text line are perpendicular to the pixel rows the result of the projection should just be a blurry mess (no clear partition of text lines (Bonus: This partition will mark the borders of the page and if the text is arranged in columns, you will get the structure of the columns).
3. Now You just need to decide which projection (step 1, or step 2) represents real text lines. You can calculate the amount of blobs (one dimensional blobs - so the processing is extremely fast) and choose the one with more blobs (there are more lines than text columns). Alternatively you can just calculate standard deviation of each projection vector and take the one with the higher 'std'. This is even much faster.
4. All the above holds if the text goes clearly in 0 degrees or 90 degrees. If it is rotated, say to 10 degrees than both projections will return a mess. In that case you can cut your document to say 5x5 pieces (25 pieces), perform steps 1,2,3 on each piece and choose the decision according to the majority.

Note: The described solution is a bit less accurate than Hough transform but it is very easy to implement, extremely fast (Entire processing is faster than just calculating derivatives of the image) + You will get for free the orientation of the text lines + partition of the document into lines & columns.


Good luck

Addition & Clarification to step 1: Explanation of step one. Suppose you have an image of width 'W' and Height 'H' and a black text on white background. By doing a horizontal projection you sum the values of pixels in each row. The result is a vector of length 'H'. Pixel Rows that don't include any parts of text (thus located between the text line) will yield a high projection values (because background is white - 255). Pixel rows that include parts of letters will yield a lower projection values. So now you have the vector of length H and you want to see if there is a clear partition of values inside it. A group of high values, than a group of low values, etc (like zebra stripes). Example: if you have 20 pixels distance between text lines and each letter has a height of 16 pixels you expect the projection vector to have 20 large values followed by 16 low numbers followed by 20 high values, 16 low, etc. Of course the document is not ideal, each letter has a different height, some have holes: (like 't' and 'q', 'i') but the general rule of partition holds. On the contrary if you rotate the document by 90 degrees and now your summation does not align with lines of text - the result vector will just have roughly random 'H' values without clear partition into groups. Now all you need to do is decide whether your result vector has a good partition or not. A quick way to do so is to calculate the standard deviation of the values. If there

is a partition - the std will be high, otherwise it will be lower. Another way is to binarize your projection vector, treat it as a new image of size 1xH, lunch connected components analysis and extract the blobs. This is very fast because the blobs are one dimensional. So the bright blobs will mark roughly the areas between text lines and the dark holes mark the text lines. If your summation was good (vector had a clear partition) - you will have few large blobs (amount of blobs ~ roughly as amount of lines and median length of a blob ~ roughly as the distance between text lines). But if your summation was wrong (document rotated by 90 degrees) - you will get many random blobs. The connected component analysis requires a bit more code (compared to std) but it can give you the locations of the lines of texts. Line 'i' will be between blob 'i' and blob 'i+1'

edited May 28 '14 at 6:53

answered May 22 '14 at 10:11

 **DanielHsH**
2,964 1 17 27

You're absolutely right, I was worried about the computation time of the Hough transform and I probably should have mentioned it. I'd like to see your solution implemented, and I might do it myself if I get a minute this morning. – beaker May 22 '14 at 13:14

@daniel, could you please write a more descriptive example of step 1? :) – clarkk May 27 '14 at 22:30

No problem I will edit my answer – DanielHsH May 28 '14 at 6:37

Are there any tools available to do this operation? – clarkk May 28 '14 at 10:17

Clarkk - what do you mean tools? You are programming in a computer language. Probably C++/Java/Phyton. Calculation of projection is done using the for() loop. Calculation of std is done also using for() loop. After you get the image as an array of gray level pixels and before you pass it to OCR you should write a code that does the processing I described. The same holds for hough transform. openCV has interfaces for almost every major programming language – DanielHsH May 28 '14 at 12:57

You can use the [Hough Transform](#) to detect the longest lines in your image and then find the predominant slope of those lines. If the slope is close to zero, your text is horizontal; if it's close to infinity, your text is vertical.

You don't mention if you're using a library to do this, but in OpenCV you could use [HoughLinesP](#). I used this [tutorial](#) on an image found on [wikimedia](#):



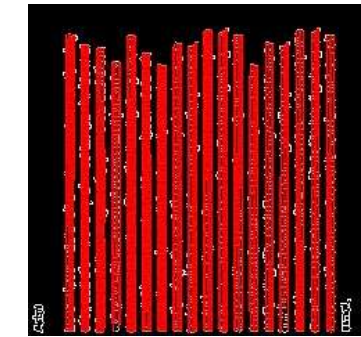
to get this image:



Then I rotated the original image:



to get this:



Since you're only interested in horizontal or vertical, you can just test whether the difference in the x-coordinates of the line endpoints is close to zero (vertical) or the difference in the y-coordinates is close to zero (horizontal).

answered May 21 '14 at 19:54



beaker

9,822 2 19 35