TNM087 – Image Processing and Analysis
Lab 4

TASK 2 - MATLAB code for counting the number of brick rows in a brick wall image

In this task you are given images of brick walls that might be rotated. You are supposed to write a code that counts the number of rows of bricks in the input image. Therefore, your code **at the first step** has to rotate the images back to the horizontal level. **At the second step**, your code is supposed to find/calculate the number of rows of bricks. Use the given MATLAB template $CountBrickRows.m$. As the input you have the input color image. Your code generates two outputs, the first one is the input image being rotated back to the horizontal level and the second one is the number of brick rows in the input image.

First step: rotating the input image to the horizontal level

The images of brick walls, usually, contain rows of mortar joints between rows of bricks. By using Hough transform, you can find the direction of the dominant lines, by which you can find the rotation angle to rotate the images back to the horizontal level. Make sure that you first understand the task regarding Hough transform in the preparation task for this lab.

We recommend you to follow the following steps to do this task:

1. Your input image is a **color** image scaled on $[0,1]$, make it grayscale, by for example using **the most appropriate** channel of its three color channels (Red, Green or Blue). **TIPS:** Look at the three channels of the test images (for example $brick1.jpg$), and choose the channel that has best contrast between the bricks and the mortar joints. This will make it easier to separate the bricks from the mortar joints in the following steps.
2. One of the first steps in segmentation, as discussed in the course, is smoothing. Therefore, smooth your image (lowpass filter) by for example a 5×5 box kernel. Make sure that your choice of the kernel size works for all test images; otherwise use a larger filter kernel.
3. One useful step here is to separate the bricks from the mortar joints, which can be done by thresholding the image using for example Otsu's algorithm. The MATLAB function $graythresh$ can be used to find a proper threshold using Otsu's algorithm. Threshold your image by this threshold.
4. Use Hough transform to find the direction of the principal lines. The angle θ corresponding to the maximum of the Hough transform, can be used to find the rotation angle to rotate the image back to the horizontal level. In the preparation task, you have all necessary information on how to find the angle corresponding to the principal lines, and how to find the **clockwise** or **counterclockwise** rotation angle from that.
5. Rotate the image to the horizontal level.

Before doing the second step of the task, test your code on the following **three** test images, $brick1.jpg$, $brick2.jpg$ and $brick3.jpg$.

Your code has to work for all of these three test images. Just notice that, the image $brick1$ is not rotated at all and your code should not change it. Make sure that the rotation angle you find for this image is 0 and not 180°. For the rest of the test images, your code is supposed to be able to rotate them to the horizontal level by the **smallest possible angle**, either by a clockwise or a counterclockwise rotation.

Second step: finding the number of rows of bricks

We suggest the following steps to find the number of rows of bricks.

1. Rotate the thresholded image (obtained in Step 3 above) to the horizontal level. Since we want the rotated image to be binary as well, don't use $bicubic$ interpolation. Use $nearest$ interpolation instead. View this image. **(Remove/comment that row later before you submit your file)**
2. As you see in the rotated thresholded image, the mortar joints (mostly) get pixel value 1 and the bricks (mostly) get pixel value 0. The segmentation is not perfect but acceptable. Now, if we take the sum of pixel values in each row, the sums belonging to mortar joints will have larger values than brick rows. Why? Calculate the sum of pixel values in each row. This can be done by the MATLAB function sum. Notice, however, that sum by default returns the sum of pixel values in each column, and what you need is the sum for each row! **It is appropriate to normalize** the sum by the length of each row, i.e. the number of columns of the input image. View your result by plotting it, using $plot$. **(Remove/comment that row later before you submit your file)**
3. As can be seen in the above plot, there are a number of peaks, corresponding to the rows of mortar joints. If a row was perfectly white, the plot would have the value of 1, (why)? Since the bricks and mortar joints are not perfectly segmented, there is no such a perfect row. Therefore, we need to decide a threshold value to define a row of mortar joint. Look at the plot to select a proper threshold, including all the relevant peaks. (Hint: since '$brick3.jpg$' is the most difficult image, use this image to select your threshold value, to make it general enough to work for all three images). After thresholding, plot your result to make sure that all rows defined as rows of mortar joint have the value 1, otherwise 0. **(Remove/comment that row later before you submit your file)**
4. In order to find the number of rows, you can, for example, count the number of times the above graph goes from 0 to 1. There are many different ways doing that. One way is to look at the first derivative of this array, by for example **correlating** it by the filter kernel $[-1,1]$. What is the filtering result every time the array goes from 0 to 1? This can help you to calculate the number of rows.
5. In order to find the number of brick rows, you need to subtract 1 from your result above, why?

Test your code using the following **three** test images, $brick1.jpg$, $brick2.jpg$ and $brick3.jpg$. Your code should find the number of brick rows in all three images correctly, which will be (as you also see in these images) 12, 10, and 10, respectively.