Instruction of Python codes to create lines from an image

The process of extracting lines from an image is divided into several sections. The sequence plays an important role in the first pass for an image, since the respective sections build on the results of the previous steps. The individual steps are:

- Rectification of the image
- Creation of a binary image
- Detect neighborhoods
- Detecting lines of the outer edges
- Filtering of the generated lines

Rectification

First, the image must be rectified by control points so that the ratios are correct. To do this, the coordinates of these points in the image space and in the space to be transformed are measured and saved in two text files in CSV format. The file name plays an important role, otherwise the files will not be found. Therefore assigned name consists on the one hand of the image name and on the other hand, separated by an underscore from the image name each "mp" for the image coordinates and "fp" for the space coordinates. The file type is to be selected as text file.

The reading of these files is defined in the code "pointsread" as a method, only the file name and path must be specified. In the code "rectify" the image is rectified in the method "rectify". Four parameters are needed for this, the image name, the file path and additionally two parameters to crop the rectified image. Especially the last two parameters have to be set for each image and it has proven to be difficult to calculate this in advance, here only "try and error" helps. As a result, the rectified image is returned and saved in a specially created folder.

Binary image

The next step is to transform the rectified image into a binary image, a binary image consists of only two color values. This is done in the method "binary" in the code with the same name. For this only three parameters are needed, the image name, the file path and a boolean variable is used, which indicates whether the image has been rectified or not. Depending on the last parameter, a different image file is converted.

The respective image is read in and then the colors are filtered out pixel by pixel and finally this black and white image is converted to a binary image over a specified threshold. The binary image is returned and stored in a separate folder.

Neighborhoods

The neighborhoods from the binary image are created for only one of the pixel colors. The nearest eight neighbors of each pixel are checked for the same pixel color, and the individual neighborhood positions are numbered (see Fig. 1).

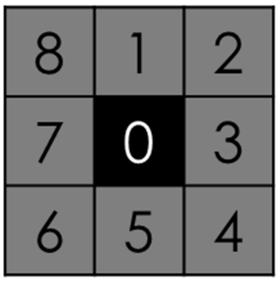

Figure 1 Neighborhood Positions

Once a pixel is found that has the color value being searched for, a nine-digit binary code is placed at the pixel location, this initially only has a one at the first of the nine digits, while the rest are set to zero. If one of the eight neighbors has the same color value, the zero is converted to a one at the corresponding position in the binary code.

Table 1 Partitioning of the binary code

1	0	0	1	0	1	0	0	1
	1	2	3	4	5	6	7	8
Placeholder	Neighborhood Positions							

If a pixel has three pixels of the same color at positions 3, 5, and 8, then the binary code looks like this: 100101001. Table 1 shows this code in its individual components. On the one hand, the first position of each binary code is only a placeholder, so that the following neighborhood positions are always stored at the same position. The neighborhoods are stored in ascending order, so that at the second position of the binary code the neighborhood relation to the neighbor one is mapped. In pixels with a different color value, only a zero is inserted. Furthermore, only pixels that have at least one pixel of a different color in their neighborhood, i.e. a boundary pixel, are binary coded. Furthermore, only pixels that have at least one pixel of a different color in their neighborhood, i.e. a boundary pixel, are binary coded.

Lines of the outer edges

In the code "vectorize" method is used to create lines from the binary code dataset. Only the image name and file path are needed. The binary code dataset will be input and the size will be taken, furthermore now two lists will be created, one for the filtered lines and one that will not be filtered. Now the pixels are queried from left to right, top to bottom for their binary code. If a pixel contains a neighborhood relationship, the code is broken down into its constituent parts so that the individual positions can be examined. Since, as mentioned from left to right and from top to bottom is worked, only the positions 3-6 are examined, which has the consequence that lines are not drawn several times.

The four positions are examined and if a neighborhood to a same-colored on this position occurs first the beginning coordinate is stored and searched for the end coordinate, for its different methods are iteratively used depending upon position. Basically, these methods are structured the same, first the pixel of the neighborhood position is examined. Here it is examined only whether also here a binary code is present, which looks then only after the same position. If there is also a positive neighbor relation here, the same method is called. After the iteration, the binary code of this pixel is then changed so that there is no longer a one at the position, but a zero. Thus, no overlapping lines are created. If the value at the given position is not positive, the pixel coordinate is set as the end coordinate. On the other hand, if there is no binary code or the edge of the image is reached, the previous pixel coordinate is set as the end coordinate.

If the end coordinate is found, the start and end coordinate, the length and the direction of the lines are stored. Here, in addition, a rough filtering (longer than 1 pixel with horizontal and vertical and √2 pixels with diagonal) of the lines is accomplished. The two line lists are output and saved.