An Implementation of a Novel A* Path Planning Algorithm for Line Segmentation of Handwritten Documents

Saverio Meucci
saverio.meucci@stud.unifi.it

Abstract

In this paper it is presented an implementation of a method for line segmentation of handwritten documents proposed by [1]. In particular, the novelty of this procedure is based on the usage of the A path-planning algorithm, modified for this task thanks to a smart combination of cost functions that allows the agent to compute paths separating two subsequent text field. Some performance measures have been computed, comparing the experimental results on two different datasets. Details about the implementation are also presented.*

1. Introduction

Current search engines provide a useful tool to search and acquire information from the Internet. However, a lot of this information is not accessible by current search engines, that can only provide information that is computerized. This not accessible information is represented in the form of scanned documents, especially handwritten documents, that cannot be understood by current search engine technology [1]. In order to make this information available, the scanned document need to be processed and its content recognized to make it searchable with search engines tailored to this task.

In this paper, an implementation of a novel line segmentation method is discussed. Line segmentation is one of the first technique that need to be applied to a scanned handwritten document, so that later each individual words can be extracted and processed. Line segmentation is a complicated task because it has to deal with many difficulties such as curved lines or partially overlapped lines; due to these difficulties, currently there is not an optimal method.

The method for line segmentation of a handwritten document, developed by [1] works in three phases. First, the document need to be binarized, thus obtaining a black and white image of the original document. Then, the text lines needs to be automatically localized inside the document, in the form of starting and ending points. Finally, using these starting and ending points for each line, a path-planning algorithm is executed to determine the path that separates each pair of text-lines.

In the following sections, details about these three phases for the implemented line segmentation method are given and discussed. A specific focus has been given to the A* path-planning algorithm that was tailored for this task. In fact, the A* algorithm, using a combination of cost functions, is able to determine paths that can cross trough obstacles to deal with overlapping text-lines.

The line segmentation algorithm have been implement in C++11 with OpenCV 3.0 to handle images and using Eclipse Luna as IDE.

2. Datasets

The datasets used to evaluate the performance of the implemented line segmentation method are two.

The *Saint Gall* dataset is a collection of handwritten documents of the 9th century, written in Latin, and contains 60 pages. Each page is written with a single column layout and some of them contains graphics [1]. The images representing the pages have a standard size of 1460×1860.

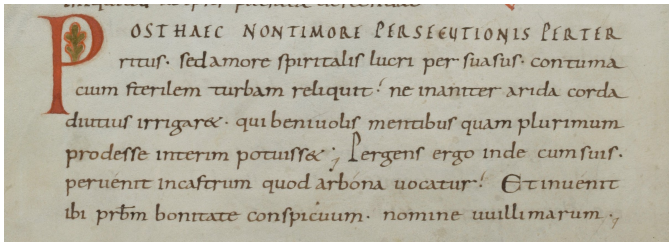


Fig 1: An example of handwritten document from the Saint Gall dataset.

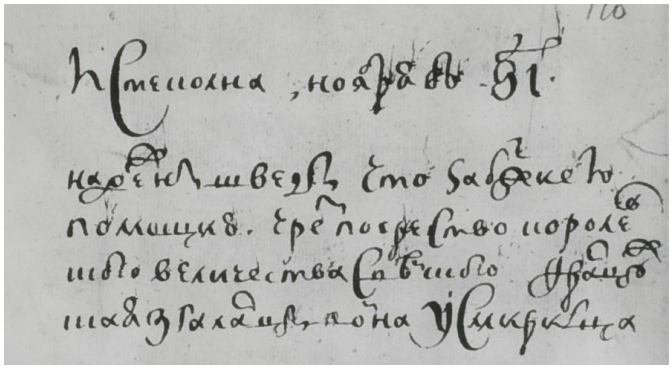
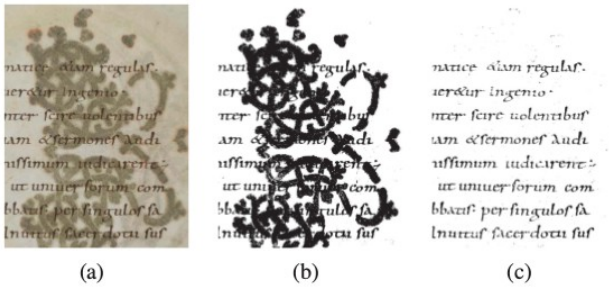


Fig 2: An example of handwritten document from the MLS dataset.

The second dataset used is a collection of 10 documents belonging to the *MONK* line segmentation (*MLS*) dataset.

The *MLS* dataset contains medieval, historical and contemporary manuscripts that has the purpose of testing line segmentation algorithms [1]. In fact, this collection of documents presents several common problems occurring in handwritten recognition such as lines with overlapping ascendants and/or descenders, scans with a light rotation and curved lines. In this case, the pages do not have a standard size.

3. Sauvola's Binarization Algorithm



(a) (b) (c)

Fig 3: Comparison of the results on a handwritten document with complicated background noise (a), using Otsu's algorithm (b) and Sauvola's algorithm (c).

In the first phase the scanned handwritten documents are binarized using a binarization algorithm. One of the most popular binarization technique is Otsu's algorithm, that is a global technique, meaning that it find an optimal threshold to be applied to the entire image. Global techniques, thus, do not take into account differences inside the image and are not good to deal with complicated background noise. For these reasons, Otsu's algorithm does not perform well for binarizing scanned handwritten document. In fact these documents present background that is complicated, as shown in Fig. 3, and can lead to significant error in the later phases of the line segmentation method, if not correctly dealt with.

To deal with this problem, a local binarization method has been used, that is Sauvola's algorithm [2]. This algorithm works as follow. A window, typically of size 20×20, is used to scan the image with a step of one pixel. For each window, the algorithm computes the mean and the standard deviation of that neighborhood of pixels. We then obtain two matrices, one for the mean values, the other for the standard deviation values, the same size as the original images. Each value of these two matrices refers to the mean and the standard deviation of the pixel that was the center of the window a that specific step.

Thanks to these values an individual threshold for each pixel is computed, as shown by Eq. (1),

$$T(x,y) = m(x,y) \cdot \left[1 + k \cdot \left(\frac{s(x,y)}{R} - 1 \right) \right] \qquad (1)$$

where $m(x,y)$ and $s(x,y)$ are the mean and standard deviation values, respectively, computed at that pixel location, k is a constant of positive values and R is the dynamic range of the standard deviation.

Using the computed thresholds (1), the image can be binarized taking into account the local differences in intensity of the image, thus leading to a better results.

This algorithm was implemented following the work of [3]. The function takes as parameters an input and an output image, the window size, set to 20×20, the R value, set to 128, and the k value set to 0.4.

In order to optimize the performance of this binarization method, we compute the integral image and the square integral image of the document image, so that the window area for each pixel can be computed efficiently. Then, diving each window area by the number of pixels in the window gives us the mean and the standard deviation for each pixel of the image with which compute the threshold values.

4. Line Localization

The second phase concerns the automatic detection of the locations of the lines inside a handwritten document. To find these locations, the projection profile analysis has been used. This method computes the horizontal ink density histogram of an image, representing the document, by counting the number of black pixels in the corresponding row. These values are then stored into a vector. We consider as starting points for line segmentation the points that fall between two local maxima of the histogram. To avoid false local maxima, a persistence threshold has been used, in order to select only the local maxima that are larger than $(\mu_h - \sigma_h)$, where μ_h is the mean of the ink histogram and σ_h is its standard deviation. These selected local maxima represented the locations of the text-lines within the document and the starting and ending points for the line segmentation are set in the middle of two subsequent local maxima.

In our implementation, first the binarized image is inverted and then its pixel values are divided by the max value, which is 255. This way we obtain a matrix of values equal to 0 or 1, where 0 represents a white pixel and 1 represents a black pixel, thus making the task of counting the number of black pixels in each row easier, since we only need to sum the values of each row. A variation of the procedure uses a binarized image that has undergone a morphological transformation such as opening, to give more weight to the text lines.

Since the threshold values depend entirely by the computed ink density histograms, the function takes as parameters only the binarized image. To compute the local maxima, an external library has been used, namely persistance1D [4], a C++ library for peak detection in one dimensional vector such as our horizontal ink density histogram.

5. A* Path-finding Algorithm

The A* is a path-planning algorithm that minimizes the sum of the costs of the path between a starting node and an ending node that are situated in a map, viewed as a graph. In our case, the graph is the binarized image and each pixel is a node. For each line in the document, the starting node is the starting point computed by the line localization phase, taken at the first column of the image, and the ending node is the same starting point but taken at the last column of the image.

The A* algorithm works as follows. The algorithm has an open set, that is the set of nodes to be explored, and a close set, the set of nodes already explored. Starting from the starting node, the algorithm takes the neighbors of the current node (that is the starting node in the first iteration) and for each one it computes a score, that is the sum of the cost of the path to the current node, the cost of the move from the current node to that specific neighbor and a heuristic that estimates the cost between that neighbor and the ending node. Each of these neighbors are then put in the open set and the current node is put in the close set. In the next iteration of the algorithm, a new current node is selected from the open set, by choosing the one that has the lowest associated scored, and the algorithm is repeated. When the current node is the ending node, the algorithm terminates.

The heuristic function needs to be admissible to achieve optimality; however, we can use a non-admissible heuristic in order to speed up the search for a path, when the optimality is not a strict requirement, as it is in our case. This can be obtained simply by multiplying by a factor an admissible heuristic such as the euclidean distance.

The specialty of the A* algorithm proposed by [1] consists in its five costs functions, discussed later, that are tailored for the line segmentation task for dealing with difficulties such as overlapping text-lines; the algorithm can, in fact, decides, at a cost, to cross through obstacles, a feature not allowed by the standard algorithm, resolving the problem with unreachable goal states.

5.1 Cost Functions

The authors [1] of this novel A* path-planning algorithm created five cost functions. These functions, combined together, compute the cost from a node to the next one. The five cost function are the following.

1) The Ink Distance Cost Functions D(n) and D(n)²:
These functions compute the distance between the current node and the closest black pixel, in the upward and downward direction, and control that the computed path stays more or less in the middle between two subsequent text-lines.

$$D(n) = \frac{1}{1+min(d(n,n_{y_u}),d(n,n_{y_d}))} \quad (2)$$

$$D(n)^2 = \frac{1}{1+min(d(n,n_{y_u}),d(n,n_{y_d}))^2} \quad (3)$$

2) The Map-Obstacle Cost Function M(n):
The function gives a penalty when the path passes through an obstacle. It allows the agent of the algorithm to cross obstacles but also discourages it when it is not necessary. The function returns a value of 1, if the agent lies on a node that is a black pixel and therefore an obstacle, it returns 0 otherwise.

3) The Vertical Cost Function V(n):
This function controls that the computed path does not deviate too much from the y-position of the starting node and ending node.

$$V(n) = abs(y_y - n_y^{start}) \quad (4)$$

4) The Neighbor Cost Function N(s_i, s_j):
This cost function is the same as the standard A* algorithm. It determines the cost of a move from the current node to a neighbor node. In our case, the agent can move in eight directions; the cost is 10 for horizontal and vertical movements and 14 for diagonal movements

These cost functions are then combined as follows:

$$C(s_i,s_j) = c_d D(s_i) + c_{d2} D(s_i)^2 + c_m M(s_i) + c_v V(s_i) + c_n N(s_i,s_j) \quad (5)$$

where c_d, c_{d2}, c_m, c_v and c_n are parameters that have been tuned by the authors of the original paper. These parameters assume different values for each dataset used. For the *Saint Gall* dataset: $c_d = 150$, $c_{d2} = 50$, $c_m = 50$, $c_v = 3$, and $c_n = 1$. For the *MLS* dataset: $c_d = 130$, $c_{d2} = 0$, $c_m = 50$, $c_v = 2.5$, and $c_n = 1$.

5.2 Implementation

The A* path-planning algorithm is executed for each line of a document found during the line localization phase. After a document is processed, the paths that separates each text-line are obtained. Each path is then used to segment the text-lines and each line saved in a separated image. Therefore, for each document image, we will have a set of binarized image, the same size as the original, each containing one single detected text-line.

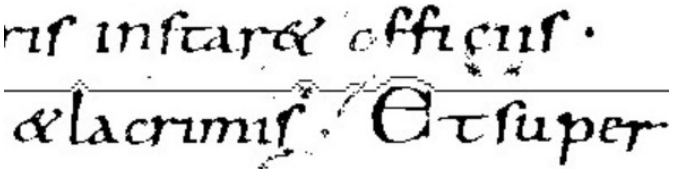


Fig 4: An example of a computed path separating text-lines on a document from the Saint Gall dataset.

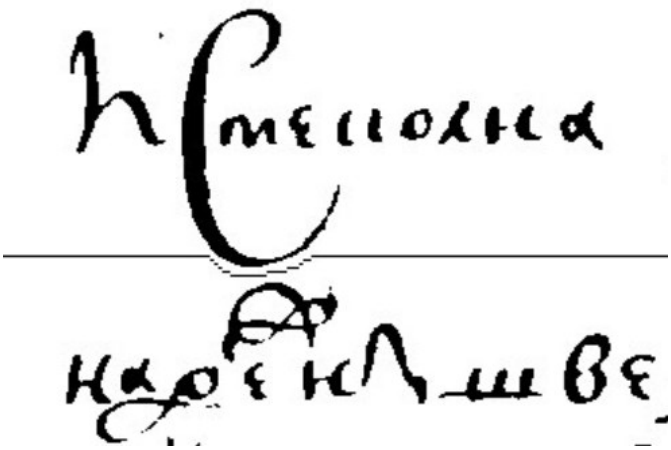


Fig 5: An example of a computed path separating text-lines on a document from the MLS dataset.

The implemented algorithm is quite slow for large images, due to the high number of nodes that need to be explored. In order to speed-up the computational time some expedients have been put in practice, that do not sacrifice too much the accuracy of the line segmentation method.

The first expedient regards how the neighbors of the current node are explored. The standard approach, making use of 8-directional movements, is to selected the neighbors within a range of one move for each direction. To boost the performance of the A* algorithm, a step of two moves have been used, meaning that are considered neighbors the nodes that have a distance of two moves, for each direction, from the current node. This approach greatly reduces the number of explored nodes, that are the pixels of an image.

Another way to improve the performance of the algorithm is to use a non-admissible heuristic function to estimate the distance between a node and the ending node. By default, the algorithm uses as heuristic function the euclidean distance, multiplied by a factor of 10, since that is the cost for a horizontal move. That represents the an ideal paths with the lowest sum of costs, which is a straight line from the starting node to the ending node. The factor is halved when a step of two moves is used to explore the neighbors; that is because, since the agent navigates the map with two steps at a time, the cost for each horizontal move become 20. Simply increasing the multiplication factor can significantly improve the speed of the algorithm.

Other precaution have been used to improve the computational time such as using a priority queue as a data structure for the open set and using a modified distance transform to compute the distance to the closest black pixel in the upward and downward direction for each pixel, improving the speed for the two ink distance cost functions.

6. Ground-truth Creation

The ground-truth for the line segmentation is created both manually and thanks to a script. In general, for each line of a handwritten document we need to create an image, the same size as the original document, containing only that binarized line; since this is the same format with which the detected lines are created, we can easily confront them and decide if the line were correctly detected.

For the *Saint Gall* dataset, information about the location of the text lines were provided in the form of XML files. These files contain the coordinates of the contour of every line in the *Saint Gall*'s documents. A python script was built in order to parse these XML files: the documents were resized to half a size, binarized and finally segmented accordingly to the contours and then saved, obtaining the format described before.

For the *MLS* dataset, no information about the lines were given. In this case, the ground-truth lines were acquired manually with the help of an image editor such as GIMP. Since there is a large variety of handwritten documents in this collection, the time needed to manually create the ground-truth varies a lot; on average, it takes around 20-30 minutes per document.

7. Results

In order to evaluate the performance of the line segmentation method, three measures have been used.

The first measure used is the pixel-level hit rate, computed as follows:

$$Hr = \frac{G(S)}{|GT \cup R|} \tag{6}$$

where GT is the set of black pixels in the ground-truth line, R is the total number of black pixels in the line detected by the algorithm and $G(S)$ is the number of shared black pixels between a particular pair of ground-truth line and detected line; basically, it computes the percentage of shared black pixels between a ground-truth line and a detected line.

At this point we have M ground-truth lines and N detected lines. The pixel-level hit rate is computed for each pair of ground-truth lines and detected lines, thus obtaining a $M \times N$ matrix P, where P_{ij} is the percentage of shared black pixels between the i^{th} ground-truth line and the j^{th} detected line. For every ground-truth line the optimal assignment is selected. The corresponding value represents the pixel-level hit rate between the ground-truth line and the optimal detected line.

Selecting the optimal assignment for each ground-truth line is not enough to determine if a line has been correctly segmented. We also need to use the text-line detection accuracy measure to do so. A line I is correctly detected if

$$\begin{cases} \frac{G_{ij}(S_{max})}{GT_i} \geq 0.9 \\ \frac{G_{ij}(S_{max})}{R_j} \geq 0.9 \end{cases} \tag{7}$$

where $G_{ij}(S_{max})$ is the pixel-level hit rate of the optimal assignment, GT_i is the number of black pixels in the i^{th} ground-truth line and R_j is the number of black pixels in the

j^{th} detected line. For every document image processed by the line segmentation algorithm we obtain these measures and the number of correctly detected lines with regards to the total number of ground-truth lines, thus with some simple operation we can obtain the total statistics about an entire dataset.

Datasets	Hit-rate	Line accuracy	Correctly Detected	Time
Saint Gall	94,08 %	97,03 %	93,57 %	20 min
MLS	85,2 %	91,7 %	72 %	15 min

Tab 1: Multiplication factor: 5, step: 2.

Datasets	Hit-rate	Line accuracy	Correctly Detected	Time
Saint Gall	94,10 %	97,6 %	93,57 %	10 min
MLS	85,3 %	91,7 %	72 %	5 min

Tab 2: Multiplication factor: 10, step: 2.

Datasets	Hit-rate	Line accuracy	Correctly detected	Time
Saint Gall	94,13 %	96,97 %	93,64 %	40 min
MLS	85,4 %	91,75 %	72 %	40 min

Tab 3: Multiplication factor: 20, step: 1.

Three different experiments have been performed changing the both the step value and the multiplication factor, using both the 60 pages of the *Saint Gall* dataset and the 10 pages of the *MLS* dataset. The manuscripts of the Saint Gall dataset contains a total of 1431 text-lines; the 10 manuscripts selected from the MLS dataset contains a total of 200 text-lines.

The first experiments, Tab. 1, uses a step value of 2 and a multiplication factor of 5. The second experiments, Tab. 2, uses a step value of 2 and a multiplication factor of 10; these changes show a significant differences in terms of computational time but not so much in accuracy. The third experiments, Tab. 3, use a step of 1 and a multiplication factor of 20; this cases is the one that gives the most accurate results, even though not by a lot, but concurrently the computational time becomes substantial. As reference, the algorithm without using any expedients, so with a step of 1 and a multiplication factor of 10, for the *Saint Gall* datasets takes around 80 minutes.

As one can see from the tables, the *MLS* dataset performs worse than the *Saint Gall* dataset; that is consistent with the fact that *MLS* is a heterogeneous collection of handwritten documents, presenting several problems that occur during handwritten recognition.

Datasets	Hit-rate	Line accuracy	Time
Saint Gall	99,8 %	99,9 %	8 min
MLS	92,8 %	90 %	26 min

Tab 4: Results obtained by the original authors of the discussed line segmentation method.

The results obtained by our implemtation are not too far from the ones achieved by the original authors of the line segmentation method [1], as shown in Tab. 4. While their r esults achieve better accuracy and lesser computational time, the results are still consistent across the two datasets used. It is to notice that [1] used the full *MLS* datasets containing a total of 995 text-lines. In this cases, statistics about the percentage of correctly detected have not been given.

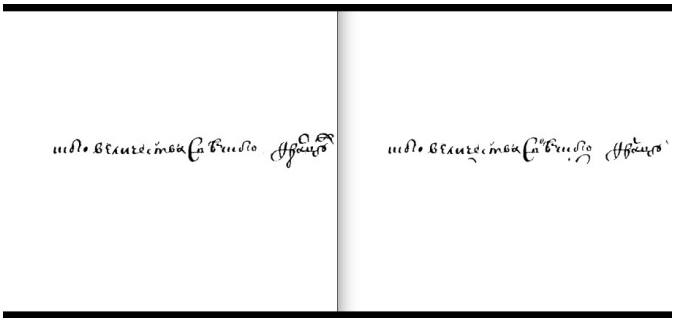


Fig 6: A detected text-line (right) from a MLS document compared with its corresponding ground-truth line (left) that the algorithm does not consider correctly detected. Hit-rate: 76,4%, Line accuracy: 87,1%. Multiplication factor: 5, step: 2.

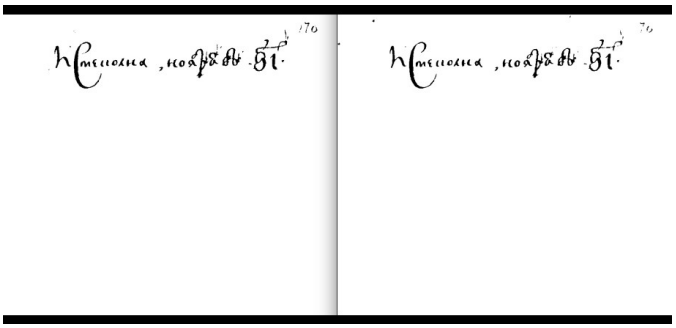


Fig 7: A detected text-line (right) from a MLS document compared with its corresponding ground-truth line (left) that the algorithm does consider correctly detected. Hit-rate: 91,5%, Line accuracy: 95,7%. Multiplication factor: 5, step: 2.

Another thing to highlight is that changing the step value to two or increasing the multiplication factor, while greatly improving the computational time of the algorithm, does not significantly alter the accuracy of the line segmentation method, thus legitimizing the usage of such expedients.

Lastly, one of the main problems when comparing the ground-truth with the detections is caused by the binarization algorithm that is not able to remove graphics from the image. A correctly segmented line may not be recognized as such because, while the text-line is correct, in the image some non-removed graphics is still present at the same y-position as the detected text-line; conversely, in the corresponding ground-truth line the graphics are correctly remove by a human user.

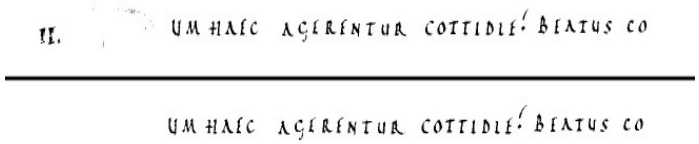


Fig 8: An example of a detected text-line (above) with some graphics not completely remove and the corresponding ground-truth line (below).

Thus, the additional noise in the detected line compared to the ground-truth line can greatly effect the measures used, as swell as missing parts of the text-line.

8. Conclusions

This paper presented an implementation of line segmentation method for handwritten documents based on [1]. First the document is binarized using the Sauvola' binarization algorithm; then the projection profile analysis is used to automatically locating starting and ending points for each line of the handwritten document; using this starting and ending points, the A* path-planning algorithm is performed to determine the path that separates each pair of lines; finally, each line is segmented and saved as a different document containing only that one binarized line, necessary to be compared with the ground-truth.

Although the experiments performed on the two datasets show good results and not too far from the ones obtained by the original authors of this novel line segmentation method, some problems still arise.

In particular, Sauvola's binarization algorithm, while better than Otsu's algorithm to deal with complicated background, is not able to correctly delete the graphics that are present in some documents. This leads to error in the line localization and consequently to the whole method, because false lines are detected. The non-text graphics can be removed by tweaking the threshold value of the algorithm but this causes problems with the binarization of the text. So a compromise needs to be found.

Moreover, projection profile analysis for the line localization is not always precise but presents some flaws. In particular, the method is not able to detect local maxima corresponding to lines that are much shorter compared to the majority of the other lines because they are below the computed threshold.

Finally, due to the size of the images representing the documents, the A* path-planning algorithm is quite slow, worse than the performance obtained by the original authors of the method, although their paper lacks references to implementation details. A considerable speed-up for the algorithm can be obtained modifying the step used to navigate the map, reducing the size of the images or using a non-admissible heuristic. These tweaks can significantly boost the speed of the A* algorithm, without effecting too much the precision of the line segmentation method.

References

[1] O. Surinka, M. Holtkamp, F. Karabaa, J.P. van Oosten, L. Schomaker and M. Wiering. A* Path Planning for Line Segmentation of Handwritten Documents, 14th International Conference on Frontiers in Handwriting recognition, 2014.

[2] J. Sauvola, M. Pietikainen. Adaptive document image binarization, Pattern Recognition 33 (2000) 225-236, 1999.

[3] C. Wolf, J.M. Jolion and F. Chassaing, Text Localization, Enhancement and Binarization in Multimedia Documents, Proceedings of the International Conference on Pattern Recognition (ICPR), volume 4, pages 1037-1040, IEEE Computer Society. August 11th-15th, 2002.

[4] Y. Kzlov, T. Weinkauf, Max Planck Institute for Informatics, Saarbrücken, https://github.com/yeara/Persistence1D.