PyFibers: A semi-automatic tool for contour extraction from cross section images of photonic crystal fibers

Anderson Mariano, Gabriela Castellano and Cristiano M. B. Cordeiro
Instituto de Física Gleb Wataghin, UNICAMP
13083-859 – Campinas – SP – Brazil
Emails: andersom.mariano@gmail.com,{gabriela,cmbc}@ifi.unicamp.br

***Abstract*—A photonic crystal fiber is a new type of optical fiber that presents an array of air holes running along its whole length and whose properties can be predicted upon precise knowledge of its cross section. Up to now, the cross section of these fibers has been estimated through manual segmentation of the contours in scanning electron microscopy (SEM) images. In this work, an image processing tool, called PyFibers, was developed, based on mathematical morphology (MM) operations. The tool extracts the contours from the images with minimal user intervention, and outputs a text file containing the contours coordinates and a DXF file with the contours. Either of these files can, subsequently, be used as input to softwares that evaluate the optical properties of the fibers.**

***Keywords*-photonic crystal fiber; microstructured optical fiber; contour extraction; mathematical morphology; scanning electron microscopy**

I. Introduction

Microstructured optical fibers, or photonic crystal fibers (PCFs) [1], [2], are optical fibers that present an array of air holes in their cross section running axially along the whole extent of the structure. The background material where the holes are inserted is usually a low loss dielectric, mainly glass (e.g. silica) or polymer. The diameter of the holes of the microstructure, as well as the mean distance between them, can vary typically between $80nm$ to $30\mu m$. Due to the high index contrast between glass (or polymer) and air, and to the fact that the structure typical dimension is within the order of magnitude of the light wavelength, the arrangement of these holes has a strong influence in the optical characteristics of the structure. Differently from traditional optical fibers, photonic crystal fibers overcome fundamental limitations imposed by the material, allowing new applications in areas such as fiber-optic communications, fiber lasers, nonlinear optics, sensors and high-power transmission.

In order to precisely predict the optical properties of such fibers, their cross section must be known with sub-micrometric, or even nanometric, precision. The contour map containing the position and shape of the microstructure can then be used in programs that numerically solve Maxwell's equations with the appropriate boundary conditions [3]. The cross section of a PCF is usually estimated through manual segmentation of the contours in scanning electron microscopy (SEM) images, which is a slow, tedious and, mainly, subjective procedure. The automation or semi-automation of the segmentation would speed up this procedure, as well as make it more objective. However, this can be difficult since the size and disposition of the holes, as well as the gray intensity and contrast of the images, may vary substantially from fiber to fiber or even from hole to hole in a single fiber image. To attempt to avoid these problems and given that the background material is dielectric, a conductive layer should be laid up over the fiber to avoid charging effects due to the incidence of an electron beam of the SEM. In several cases, however, some image degradation still occurs as, e.g., brightness gradients, particularly at the edge of the holes.

To perform the segmentation of the PCF images, a semi-automated tool, called PyFibers, was developed, based on mathematical morphology (MM) operations [4]. For this purpose the MM toolbox [5] was used, developed for the high-level, open source, programming language Python (www.python.org) [6]. Given an image of the cross section of a photonic crystal fiber, PyFibers performs the segmentation of the contours, and outputs a text file containing a list of coordinates of the contours of every hole present in the structure [7], as well as an image vector file in the DXF format. These outputs can, then, be the input of any program that evaluates the fiber optical properties. Although the image processing operations used in PyFibers are mostly well-known (most come from the field of MM), the application and implementation here proposed is original. Every step of the processing was solved in order to minimize user intervention, aiming at end users without knowledge of image processing. The tool has an easy-to-use GUI and its source code is free (can be supplied by emailing the contact author).

In short, the program goes through four main steps to achieve its goal: 1) thresholding; 2) smoothing of holes' contours; 3) contours extraction; and 4) computation of coordinates list and DXF file generation. The following sections give a detailed description of these steps.

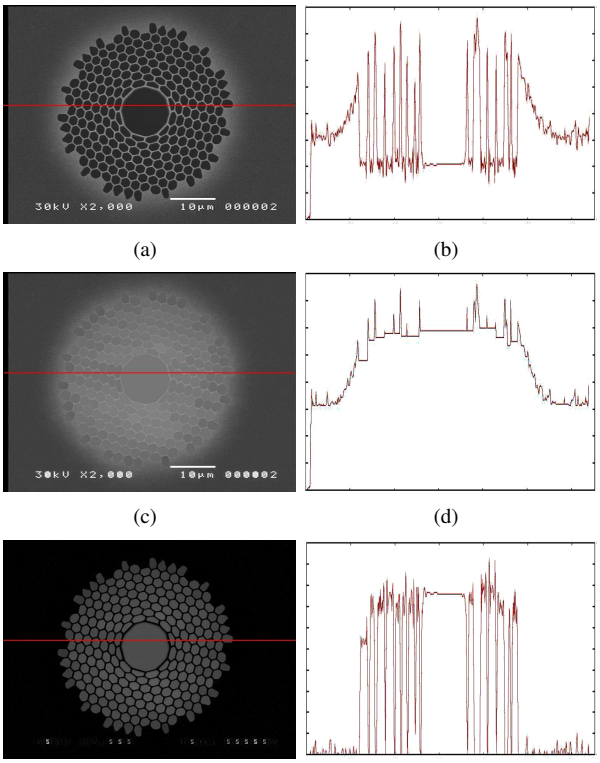


Figure 1. **(a)** Original Image. **(c)** Result of applying the operation *close holes* to the image in 1(a). **(e)** Result of subtracting the image in (c) from the original image. **(b)**, **(d)** and **(f)** Brightness profiles along the indicated line of images in (a), (c) and (e).

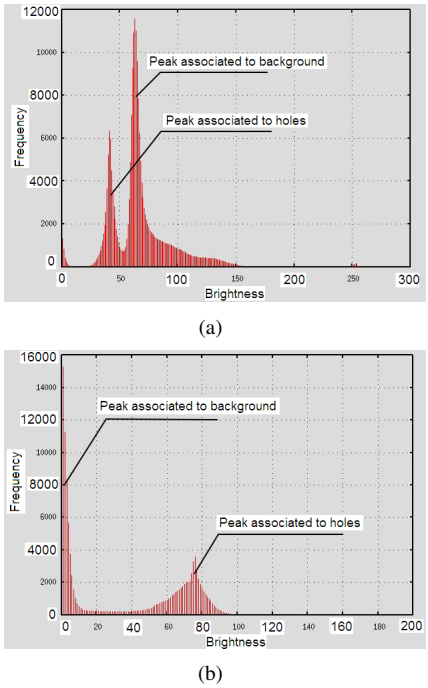


Figure 2. **(a)** and **(b)** Histograms of images in Figures 1(a) and 1(e), respectively.

II. Thresholding

From the gray level histogram of the fiber images, it can be observed that the holes always form a clear peak in it (Figure 2(a)). To chose the threshold value directly from the histogram without user intervention the well-known Otsu algorithm [8] was used, which is completely automatic. This algorithm separates the image pixels in two groups such that the covariance between them is maximal. However, the results achieved were not satisfactory, due to the presence of a smooth brightness gradient in the background region of some fibers (e.g. Figure 1(a)), and to the fact that most images also have brightness gradients at the edges of the holes. It should be stressed here that the aim is to obtain precise coordinates of the contours, and not only to separate holes, which implies on a limitation on the choice of threshold. Thus a manipulation of the gray level image was performed to improve the result of the thresholding step. The adopted strategy was to strengthen the image edges, in an attempt to make the holes flatter (i.e. make their brightness as uniform as possible), and to eliminate background clutter.

To strengthen the image edges it was sought to eliminate all background details from the image, increasing the contrast between background and holes. The algorithm that gave the best results was based on the application of the operation *close holes* to the original image. This is a morphological operator based on the *opening by reconstruction* operator [9], which fills the holes in every connected component of an image. Its main characteristic is to close holes without changing the background of the image. In this case the aim was to eliminate background details leaving only the holes in the image, and to achieve this, four steps were performed.

1) The operation *close holes* was applied to the original image, resulting in an image where all the holes were closed. Figures 1(a) and 1(c) show examples of original and processed images, and Figures 1(b) and 1(d) are their respective brightness profiles for the drawn red line.

2) The image with the closed holes was subtracted from the original image, in order to obtain just the holes. Figure 1(e) (and respective brightness profile 1(f)) shows this result, where most of the structure belonging to the image background was eliminated and a contrast gain was achieved, which drastically simplifies the choice of threshold value. This contrast gain can be easily seen in Figures 2(a) and 2(b), which show respectively the gray level histograms of the original and final images. Also, the peak associated with the image background has become much narrower, meaning that the background is more uniform.

While the described procedure gave good results for several images, other images still presented room for improvement due to the presence of edge gradients. To extra strengthen the holes' edges avoiding a critical increase in

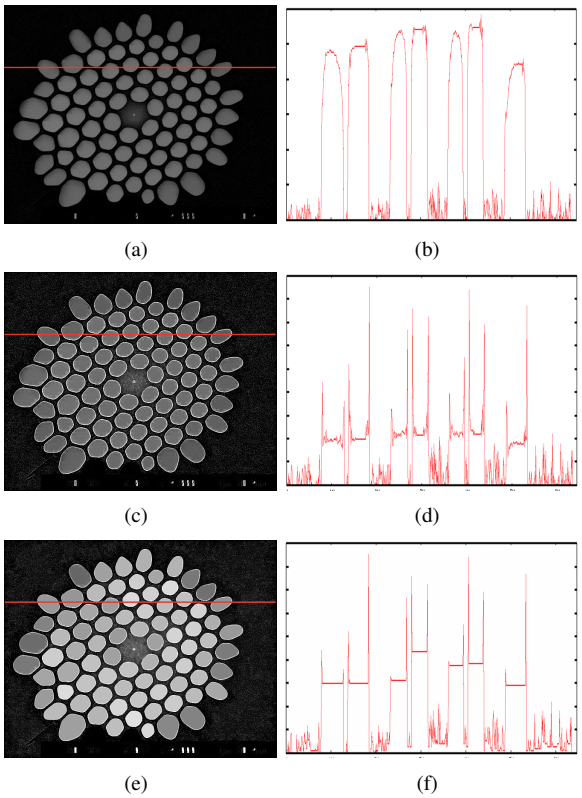


Figure 3. **(a)** Image obtained in the previous step. **(c)** Result of edge strengthening. **(e)** Result of applying the operation *close holes* to the image in (b). **(b)**, **(d)** and **(f)** Brightness profiles of (a), (c) and (e) respectively.

noise level new steps were taken.

3) The edge image was multiplied by a weighting factor (typically 4) and then it was added to the previously treated imaged, resulting in an image where the edges had a significantly larger brightness value than the rest of the hole. This makes the interior of the hole to be surrounded by bright rims. Figures 3(a) (3(b)) and 3(c) (3(d)) show examples of treated image with Steps 1 and 2, and resulting image from Step 3.

4) The operation *close holes* was applied once more. Since the holes of the image from Step 3 have brighter edges than the rest of the image pixels, this operation acts to close the interior of the holes, which will then have constant brightness. Thinking of the image as a surface, this can be interpreted as the holes becoming flat, as can be observed in Figure 3(e) (3(f)).

III. Contours' smoothing

The next required step is to smooth the contours of the holes. The simplest approach would be to apply the morphological operations *opening* and *closing* to the binary image resulting from thresholding. Using large structuring elements, small irregularities of the contour would be reduced, along with small holes and branches. While this procedure worked satisfactorily for some images, it did not work for images with holes very close together, separated by few pixels. In those images the closing operation connected the holes. Processing the holes individually, by labeling and separating them into individual images, would solve the problem, but with high computational cost. A general and cost effective solution must take into account the physical aspects of the microstructured fibers. Due to the manufacturing process [2], that is based in heating a macroscopic version of the fiber (the preform) and pulling it through a fiber drawing tower, the contour of the holes should be smooth by definition. Contours with grooves or ramifications do not have any physical meaning. The goal is then to smooth the contour while also correcting segmentation defects, preferably only acting in contour regions where these defects appear. Figure 4(a) shows an example of a specific hole and in Figure 4(b) the segmentation of this hole was added. The small defects coming from the segmentation can be better visualized in Figure 4(c) where the red circles indicate regions with pixels that should be part of the contour, but were poorly segmented. To improve this result and attempt to obtain a smoother profile the gaps in some parts of the periphery of the holes must be filled - therefore an expansive operation should be performed, but only in specific regions.

Basic MM operations (such as *erosion* and *dilation*) rely, however, on whether the structuring element (SE) fits perfectly within or outside the image. In these operations there is no tolerance with respect to this fit, and therefore they act the same way throughout the whole image. An adaptable operation, which acts differently on specific parts of the contour, was developed based on a modified version of the *erosion* operation, which consists on introducing a degree of tolerance in the fit criterion of the SE [10], [11].

A. Erosion with tolerance

To introduce a degree of tolerance to *erosion*, first recall that the *erosion* operation of an image A by a structuring element B may be defined as "the set of points of A reached by the origin of B when B is moved inside A, such that all the points of B coincide with points of A" [9]. Therefore the *erosion with tolerance* x of an image A by a structuring element B may be defined as

> "the set of points of A reached by the origin of B when B is moved inside A, such that at least $N - x$ points of B coincide with points of A",

where N is the total number of pixels of B. To understand the result of this operation and its dependence on x, see the contour of the hole shown in Figure 4(b). In the examples that follow a circular structuring element (SE) of radius 3, which can be seen in red in the figures, was used. This element has 37 pixels and therefore x can, in principle, range from 0 to 37.

Figure 5(a) shows the result of the *erosion with tolerance* operation for a value $x = 0$, i.e. without tolerance, and the

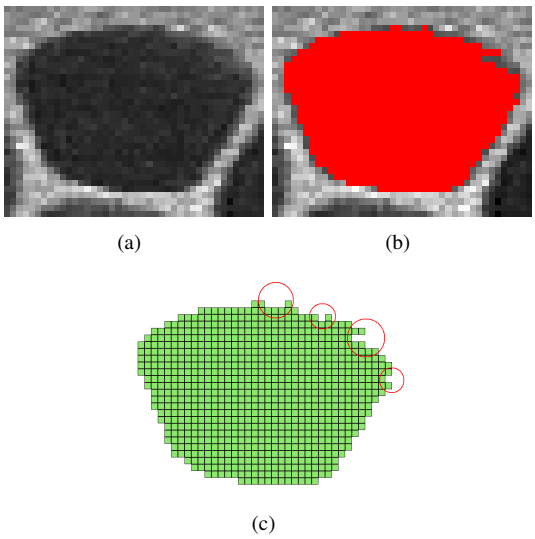


Figure 4. (a) Part of a PCF image showing one hole. (b) Result of the segmentation of this hole. (c) Illustration of the hole. The red circles indicate regions with pixels that should be part of the contour, but were poorly segmented.

result is what would be obtained from a conventional *erosion* operation. In this figure, as well as in the subsequent ones, the original image is represented by green squares, the image resulting from the operation is represented by black dots, and the SE, when present, is shown in red. In Figure 5(c) it can be seen the result of the operation with a value of $x = 1$ (tolerance of 1 pixel). The SE has thus to fit at least 36 of its pixels within the image in order for the *erosion* to be performed. It is worth noting that this does not imply an increase of 1 pixel in the whole contour of the hole, as would be obtained for example through *dilation*. Only in a few points the SE does not entirely fit the image (hole). Following the same reasoning, if $x = 2$ (2 pixels tolerance), the result seen in Figure 5(e) is obtained. The images in Figures 5(b), 5(d) and 5(f) correspond to the *dilation* of the images in Figures 5(a), 5(c) and 5(e) respectively, using the same SE. With respect to the original images, this is an analogue operation to the *morphological opening*, but with tolerance in the *erosion*. The result of this operation is, as desired, a rounder contour, where the grooves in the original contour have been successively filled up. It should be noted that the final result depends, obviously, on the SE used.

The implementation of the *erosion with tolerance* operation was achieved through convolution of the thresholded image with an image version of the SE (convolution matrix), followed by another thresholding. This is possible since the image resulting from the convolution will have brightness values ranging from zero up to the sum of all the SE pixels (case of perfect fit). Pixels with value equal to $N - x$ in the final image represent points where the SE coincides with the image in $N - x$ pixels. Application of a thresholding operation to this image, using $N - x$ as the threshold value,

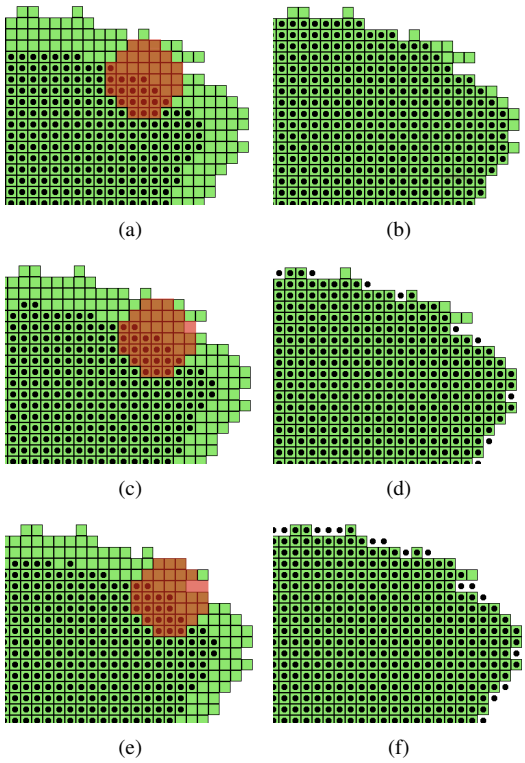


Figure 5. (a), (c), (e) Examples of applying the *erosion with tolerance* operation, with 0 pixels tolerance in (a), 1 pixel tolerance in (c), and 2 pixels tolerance in (e). In all figures green squares represent the original image, black dots represent the image resulting from the operation, and red squares (when present) represent the structuring element (SE). (b), (d), (f) Result of the dilation operation applied to the images in (a), (c), and (e) respectively, using the same SE.

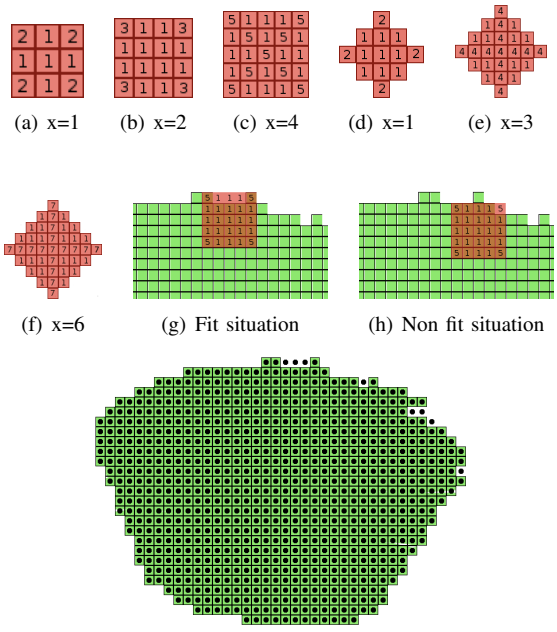
will result in a condition equivalent to the definition of the *erosion with tolerance operation* stated above.

B. Choosing the structuring element

In order to modify only the contour parts that present segmentation errors, while modifying as least as possible the remaining parts, different weights were given to the pixels in the SE (convolution matrix). Figure 6 (middle) illustrates the use of this artifice to correct errors with a specific pattern using a square SE with weight 5 in the corners (Fig. 6, left), chosen to allow a tolerance of up to 4 pixels. The top image shows a situation where the fit is accepted, since there are 3 pixels of the SE outside of the image. On the other hand, the bottom image shows a situation where the fit is not accepted, because even though only 1 pixel is outside the image, it has weight 5 and the tolerance is not respected.

Combining a set of non-flat SEs and tolerance values most of the segmentation defects found in our set of test images were corrected. The SEs used in the final algorithm are illustrated in Figure 6 (left). The result of applying this set of SEs to the contour of a hole is shown in Figure 6 (right), where it can be observed that the errors in the contours were

eliminated.



(a) x=1 (b) x=2 (c) x=4 (d) x=1 (e) x=3

(f) x=6 (g) Fit situation (h) Non fit situation

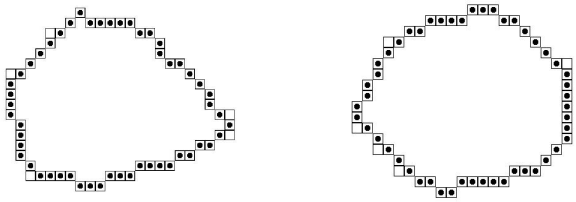
(i) Result

Figure 6. **(a)-(f)** Set of non-flat SEs used in the final algorithm with the tolerance value used for each element. **(g)** Fit situation: 3 pixels are outside of the image, which is allowed by a 4 pixels tolerance. **(h)** Non-fit situation: although only 1 pixel falls outside of the image, it has weight 5 and therefore does not respect the tolerance condition of 4 pixels. **(i)** Result of applying this SEs set to the contour of a hole. Observe that the errors in the contours were eliminated.

IV. Contours' extraction

As stated earlier, the output from PyFibers will be used in other programs that use the fiber contours to calculate the optical characteristics of the waveguide. To be as general as possible two output formats were developed: a text file containing the holes contours coordinates and a DXF file with the contours.

In the first case it is necessary to ensure that the contour line is only one pixel thick. For this, we adapted an Image Foresting Transform (IFT) [12] algorithm previously developed in [13], in order to eliminate the extra pixels. The IFT treats the image as a graph, having as input parameters the binary image with the contours, an image with seeds (in this case, a pixel from each contour), an adjacency relation (an 8-neighborhood was used) and a cost function (the geodesic distance in this case). This information is used to calculate the minimum-cost paths for each pixel in the image, starting from the seeds. The seeds that manage to "raise" pixels (lowest cost value) become roots of the image trees and each tree is labeled, defining an image region. The IFT output parameters are thus the labeled image, a cost image and a graph indicating the predecessor of each pixel in its minimum-cost path.



(a) (b)

Figure 7. Illustration of the IFT algorithm: The squares represent a contour obtained by segmentation. The circles represent the resulting contour after the IFT algorithm.

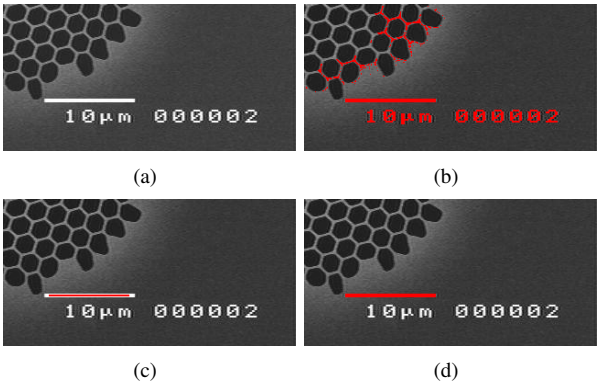
The predecessors graph generated by the transform was used to obtain the minimum-cost path to close a given contour, starting from its root and returning to it. The result is that unnecessary pixels are excluded, as it may be seen in Figure 7. The use of the geodesic distance as cost function enforces that only the pixels from the original contour (which has thickness greater than one pixel) enter the calculation. The predecessors' graph generated by the IFT results on a one pixel thick path along the contour, which is exactly what is needed. The algorithm in [13] was implemented in Python in an optimized way, so as to return only the predecessors' graph, and assuming that the input would be a binary image with only one seed per contour.

V. Computation of coordinates' list

A. Scale bar segmentation

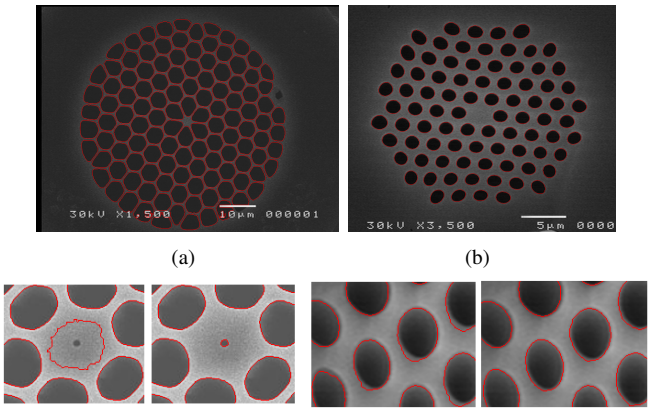
To obtain the coordinates of the contours in micrometers the scale bar needs to be segmented. This was done by first applying a threshold to the image aiming at segmenting the bar and the white letters at the bottom of the image. The Otsu algorithm [8] was again used to find the threshold value, but taking into account only the lower right corner of the image instead of the entire image. In this way the area occupied by the legend becomes significant with respect to the total area, and the threshold value that maximizes the covariance between the classes will segment the bar and the white letters, which have brightness very different from the rest of the image. Figure 8(b) shows the result of the threshold applied to 8(a).

After thresholding it is necessary to eliminate the letters and other structures that might have been segmented. For this it was used the fact that the bars have a long, rectangular and horizontal shape. Therefore an *erosion* by a horizontal SE was used, actually, a line with length based on the mean size of the holes. This was in turn obtained from the holes' area, that was measured previously. The result of this step is shown in Figure 8(c). To obtain the whole bar again, the eroded image was used as marker for the *inferior reconstruction* of the thresholded image. Figure 8(d) shows the result of the reconstruction. With the bar segmented, the corners' coordinates are obtained and this information is



(a) (b) (c) (d)

Figure 8. Example of scale bar segmentation. (a) Original image. (b) Result of thresholding (a) using Otsu algorithm in the inferior right corner of the image. (c) *Erosion* of (b) with a horizontal line. (d) *Inferior reconstruction* of (c).



(a) (b) (c)

Figure 10. (a) and (b) Result of automatic segmentation of 2 PCFs. (c) Details of automatic segmentation (left) corrected with user intervention (right) for 2 PCFs.

used to measure its length, in pixels. This parameter is used to convert the holes' contours' coordinates from pixels to micrometers - but here the user has to enter the scale value.

B. Output files

Once the holes' coordinates have been transformed to micrometers, they are saved into two formats, a text file and a DXF (AutoDesk Drawing Interchange format) file. The text file has 4 columns, containing a reference number for the hole, a sequential number for the pixel in the contour, and the coordinate values (x,y) of the pixel, in micrometers. The origin is selected by the user with the mouse. The DXF file was created using the open source SDXF package (http://pypi.python.org/pypi/SDXF/1.1).

VI. User interface

The aim of Pyfibers is to be routinely used to segment SEM images, as they are acquired. These images may be very different from each other - the program was tested in a small database and even so a few images showed holes with segmentation defects. Images that present regions with higher or lower brightness values end up with some holes being poorly segmented, as in these cases it is hard to find a threshold value that will suit the whole image. Another problem is presented by holes superposed to the scale bar, leading invariably to a faulty segmentation. In order to be practically useful the program must correct the segmentation errors, or in the worst case scenario, it must be able to exclude the holes poorly or not segmented.

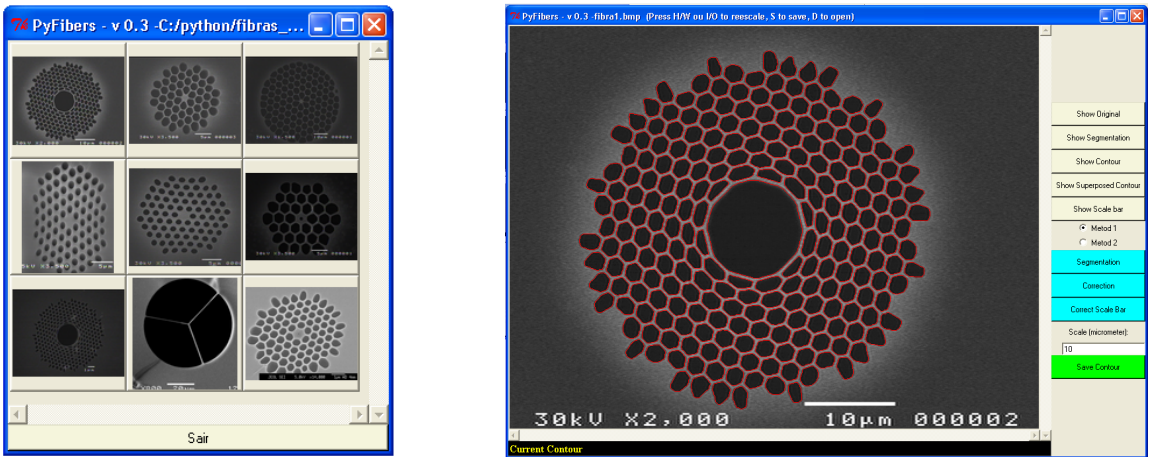
To correct these defects requires user intervention. Therefore a graphical user interface (GUI) was developed, using the TkInter module for Python (http://docs.python.org/library/tkinter.html). The GUI was based on examples found in [14]. Details of the interface may be seen on Figure 9. The program allows definition of a default directory to search for the images, or otherwise it opens a window for directory selection. The program then searches for images in the directory, creates thumbnails and uses them to generate buttons with links to the images (Figure 9(a)). By clicking on the icon the program calls another window, with image processing options. The program also allows mouse input, and includes semi-automatic (interactive) correction methods for the holes and the scale bar in cases of poor segmentation. For this, the user just clicks on the defective hole or bar, and the segmentation is repeated, but locally, isolating the hole (bar) from the rest of the image, which allows better performance of the thresholding algorithm.

VII. Conclusion

Here was presented the PyFibers semi-automatic tool for the extraction of contours of the cross section of photonic fibers in SEM images. The tool is mainly based on mathematical morphology operations. A first segmentation attempt is carried on automatically, and then the results can be corrected through user intervention when necessary. PyFibers was tested on a set of 9 different PCFs that sum a total of 985 holes, showing good results in 925 holes ($\sim 94\%$) without user intervention. Figures 10(a) and 10(b) show some results. User intervention allowed to repeat the segmentation locally in a small region around the defective holes, and it was able to increase the number of holes properly segmented to 945 ($\sim 96\%$). Figure 10(c) shows results of local correction. Figures 11(a) to (r) show the results for all the PCFs tested: in each pair of PCFs images shown, the first one is the result obtained with the automatic method, while the second one is the result of the semi-automatic method.

The main developments achieved in this work were:

1) A method for eliminating brightness gradients from the image background image based on the close holes operation. This method increases the contrast between holes and background, simplifying the choice of threshold value.



(a) (b)

Figure 9. User interface. (a) Navigation window, with buttons for every image in the directory, with links to the processing window. (b) Processing window - it has visualization area with scrolling bars, a status bar to give instructions to the user, and several buttons linking to processing routines.

Parallel operations to strengthen the edges, seeking to reduce segmentation errors, were also developed.

2) A method to eliminate small segmentation errors and smooth hole edges based on a modified *erosion* operation. This method can be applied in any situation where smoothed image contours are needed, which occurs frequently in many applications involving image processing.

3)An algorithm to segment the scale bar and convert coordinates from pixel units to micrometers.

4) A GUI to simplify access to various correction routines, to be used in case of failure of the automatic segmentation algorithm. This interface allows the user to input information through the mouse.

Although the image processing operations used in PyFibers are well-known, their application to this problem is original, since the whole field of photonic fibers is an emergent field of research.

ACKNOWLEDGMENT

We thank FAPESP-Brazil for financial support and Marcos Franco (IEAv) for helpful discussions.

REFERENCES

[1] J.C. Knight. Photonic crystal fibres. Nature 424:847-851, 2003.

[2] P.J. Russell. Photonic crystal fibers. Science 299:358-362, 2003.

[3] D. Mogilevtsev, T.A. Birks, P.J. Russell. Localized function method for modeling defect modes in 2-D photonic crystals. Journal of Lightwave Technology 17(11):2078-2081, 1999.

[4] J. Serra. Image analysis and mathematical morphology, Vol. I. Ac. Press, 1982.

[5] A.G. Silva, R.A. Lotufo, R. Machado. Toolbox of image processing for numerical Python. Proc. 14th SIBGRAPI, p. 402, 2001.

[6] G.V. Rossum. Tutorial Python, V. 2.3.2, 2005. Tech. report, Python Software Foundation. http://www.pythonbrasil.com.br/moin.cgi/DocumentacaoPython.

[7] A. Mariano, G. Castellano. Contour segmentation of the transversal section of photonic fibers in SEM images using mathematical morphology. Proc. 8th Int. Symp. Math. Morphology, p. 31-32, 2007.

[8] N. Otsu. A threshold selection method from gray-level histograms. IEEE Trans. Systems, Man, Cybern. 9(1):62-66, 1979.

[9] E.R. Dougherty, R.A. Lotufo. Hands-on Morphological Image Processing. SPIE Press, 2003.

[10] P. Maragos, R.W. Schafer. Morphological filters - Part II: Their relations to median, order-statistic, and stack filters. IEEE Transactions on Acoustics, Speech, and Signal Processing 35(8):1170-1184, 1987.

[11] P. Soille. On morphological operators based on rank filters. Pattern Recognition 35:527-535, 2002.

[12] A.X. Falcão, J. Stolfi, R.A. Lotufo. The Image Foresting Transform: Theory, algorithms, and applications. IEEE Transactions on Pattern Analysis and Machine Intelligence 26(1):19-29, 2004.

[13] G. Castellano, R.A. Lotufo, A.X. Falcão, F. Cendes. Sulcal lines extraction for cortex characterization in cerebral MRI using the IFT. Proc. 16th SIBGRAPI, p. 355-362, 2003.

[14] M. Lutz. Programming Python. 2nd ed., O'Reilly, 2003.

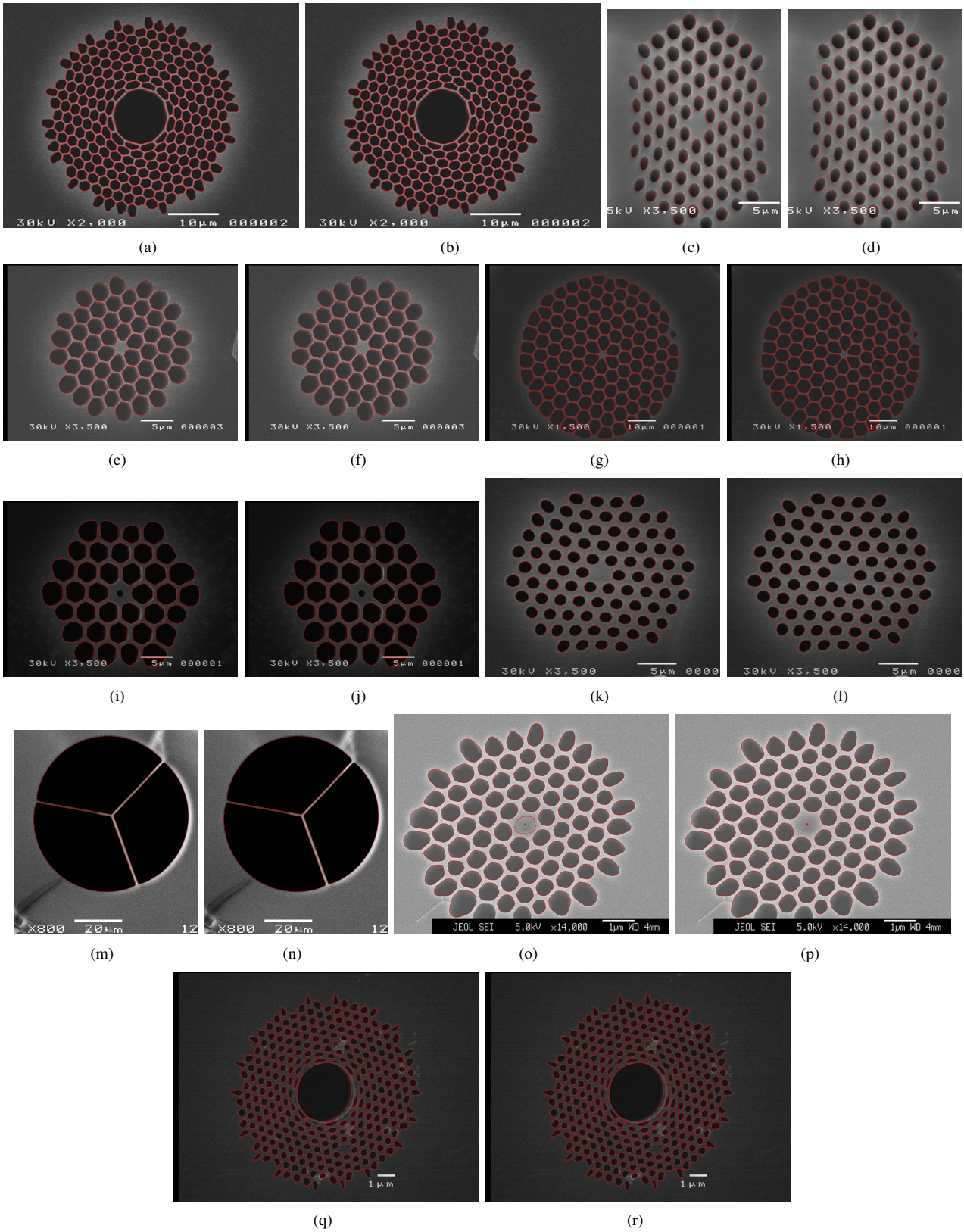


Figure 11. Comparison between the results of the automatic and semi-automatic segmentation methods. In each pair of PCFs shown, the first one is the result obtained with the automatic method, while the second one is the result of the semi-automatic method.