An image processing algorithm for fluorescent objects

Localization of Microtubles

David Zwicker[*], Felix Ruhnow[†]

Thursday 18th September, 2008

[*] Max Planck Institute of Molecular Cell Biology and Genetics, Dresden – Email: zwicker@mpi-cbg.de
[†] Max Planck Institute of Molecular Cell Biology and Genetics, Dresden – Email: ruhnow@mpi-cbg.de

Contents

1 Introduction

Microscopy of fluorescent objects is a very common way of retrieving deeper knowledge about certain processes, for instance in biology. Due to physical effects the data taken by a camera is not the exact copy of the reality. There are several effects, that distort the original data:

background light is added randomly to the whole image.

convolution of the data with the point spread function (PSF) of the optical setup, due to quantum mechanics. This is especially noticeable, if the wavelength of the light is about the size of the object.

random noise is added to all data points due to statistical processes.

An ideal processing of the microscopy images would remove all these distortions, although this is naturally not possible, because lost information cannot be retrieved. It is self-evident, that an automated program can achieve the goal only partly. This document describes an algorithm, that is intended for processing sub-wavelength objects. It is designed to run without user interaction on single images as well as on whole movies. It should return all important properties of the objects seen in the frames. The algorithm supports two different kind of objects in the images:

point-like objects occur for example if one images fluorescent nm-sized particles.

chain-like elongated objects are important for measuring long objects – in our case microtubules.

Beside these two types, nothing else may be processed by the algorithm. Instead, it tries to ignore any noise and other distracting objects. The input data has to be passed as two-dimensional gray images, where the brightness is directly proportional to the intensity. A typical example of such images is shown in Figure 1.

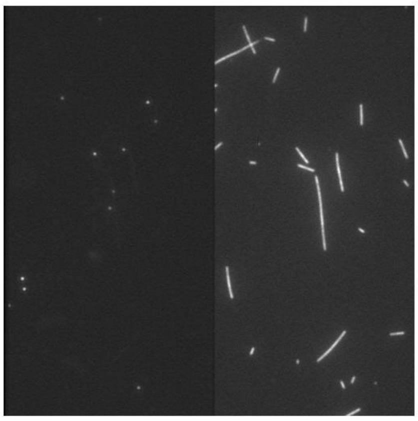
Fig. 1: *Typical microscopy image*

Further requirement on the algorithms are, that possibly all objects should be found – including objects, that are close to or even cross each other. Furthermore, it would be helpful, if important values would have an error estimation. Finally, it is a basic requirement, that the algorithm is configurable, such that one for example can increase the speed of processing or exclude certain information.

There have been quite some afford in the past, to get such an automated program. Very good results have been achieved for point like particles alone [1, 2]. For elongated objects different methods have been used:

- Fluorescent beads have been attached to microtubules to visualize the movement [3]. Although this is no direct imaging, the spacial resolution is quite high for the special application of localization a moving end point.

- The line profile (approximately) perpendicular to the microtubuls has been investigated, e.g. by determining the centroid [4] and
- by doing some more intense calculation using a Gaussian-fit to get a good estimation of the center line of the object [5]. This method cannot work at the end points, such that measuring the exact length of microtubules is not feasible.

Additionally, there have been problems with the automated initial localization step for elongated objects, which sometimes may be avoided by using a manual indication step beforehand [5]. Especially, precise length measurement of elongated objects and the combination with particles is a thing, that has not been done up to now. We try to provide a program, which comes up with these new goals.

Noticing the excellent results, that have been achieved in single particle tracking using 2D-Gaussian-fitting[1], we try to adapt this method for the more complex elongated object. The additional benefit of this approach is, that both object types may be processed in a similar way. The programming language we use is MatLab with heavy use of the Image Processing and Optimization Toolbox. In general, it should be possible to transfer the methods to any other system without too much trouble.

2 Description of the Algorithm

The algorithm handles each image of a possible movie independently. In this way, time information is completely neglected.[1]. The scanning of a image is generally done in two steps: First the objects should be found roughly through a quick scan of a binary image using a suitable threshold. The second step uses the gathered data to make up a model and use a most-likelihood method to fit certain regions of the image to increase the accuracy and precision of the data from the first step. It has been shown [2, 1], that using this way the best accuracy can be achieved for sub-wavelength objects.

2.1 Step I: Rough Scan

In the beginning, the program has to locate the objects in the image. Though this is an extremely easy task to do for any human-being, a lot of problems arise for automated computer programs. In fact, there are people getting around this by using some manual identification procedure [5]. As computers get faster, there are more and more people working on general image processing, thus hopefully leading to better recognizing in the near future. Our approach to handle the situation is to simplify the image as far as possible: The crucial part is to create a good binary image of the gray basis for the following feature detection. Binary images may have the same size as the original ones, but the intensities may only have two values: zero (black) or one (white), representing background and objects, respectively. Using this information the algorithm tries to estimate the required position data.

2.1.1 Feature detection using binary image

There are numerous methods for creating a binary image, where the ones we use are all described in the section 3.1. The essential property of this image is the highlighting of the

[1] However, the FOTS application uses the (position,time)-coordinates to connect objects after the scanning.

regions, where objects are assumed. It is therefore used to to get a first idea of the content of the original image. Using some region evaluation function, we find all connected regions and analyze them separately in the following, because by the way the binary image is created it is assured, that objects in one region do not interfere with some objects of other ones.

As we assume there are mostly point-like and elongated objects, they are easily distinguished by their area. There is an option in the settings, to influence this distinction, because the area naturally depends on the size of the objects. Additionally there are options to tell the algorithm to look only for one type of objects, where the above distinction is not necessary. Depending on the decision the next steps differ depending on the type of object, which is assumed to lie in a certain region:

point-like objects are easily handled, because only the center of the point-like object has to be determined. It is enough to take the centroid of the binary image, because the data will be enhanced in the second step anyways.

In certain situations it might happen, that two point-like objects are so close, that their areas in the binary representation overlap. In such cases the algorithm has the ability to take more then one center points in one binary area. The maximum number of points to find may be set in the configuration, such that the program may be forced to find exactly one object per region. Otherwise, the non-connected local maxima of the gray image restricted to the current region are found and taken to be as center points.

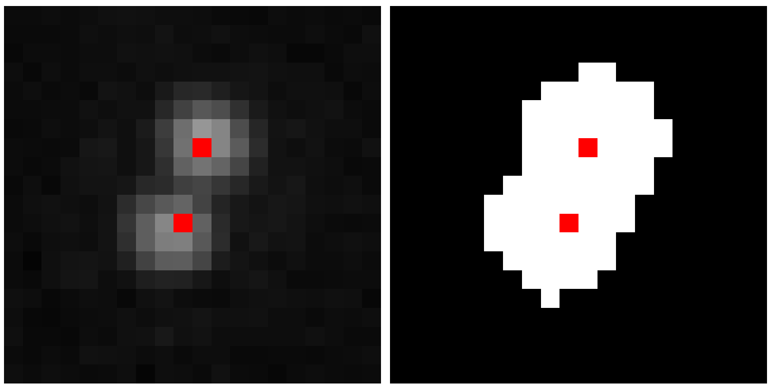

Fig. 2: *Two close-lying particles (left) and the thresholded binary image (right). The local maxima are marked in red.*

The whole situation is a bit more complicated for elongated objects:

elongated objects are theoretically described by a smooth curve of finite length. There is no way to represent this exactly in computer arithmetics. In contrast, we are satisfied with an ordered finite list of points, which approximates the center line of the elongated object.

Such lines may be found using a thinning function [6] applied to the binary image. Thinning removes points at the border of the white areas until thin lines remain. The next evaluation has to find those lists of successive points. An additional difficulty arises, when two elongated objects cross each other, thus ending up in the same region of the binary image.

To solve this problem, the algorithm counts the neighboring white pixels of every white pixel in the thinned image. The number of surrounding pixels has to lie between one and eight, having different meanings in different cases, leading to an easy distinction:

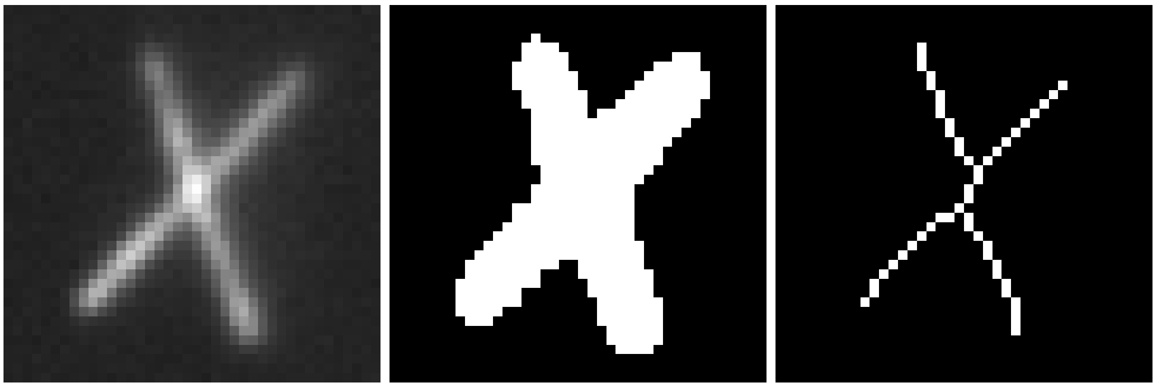

Fig. 3: *Two crossing microtubules in original, binary and thinned representation (f.l.t.r.)*

Pixels	**Meaning**
1	end of chain – there is only one direction, where the chain may go on.
2	middle part of chain – there are exactly two directions and therefore only one way of a meaningful connection.
more	this has to be a crossing of chains, as there are three or more directions for continuing chains. Additional decisions have to be made. Though due to the thinning procedure it might happen, that there appear two short chains at an endpoint of an elongated object, although there should only be one. One has to rule out such events as well.

An additional choice has to be made in the case of crossings: Because more than two chunks are entering the crossing point, it is not easy to decide which chunks belong to each other. The algorithm uses some rules considering mainly the angle of incident, but it is certain, that it is not always perfect, as it is sometimes even by eye hard to judge how the objects cross.

The next step would be to start at an end point and put neighboring pixels successively to a list. This has to be done until all pixels in the image belong to an object. The result are as many lists of points as there are objects in the region. Each list gives an approximation to the center line of such an elongated object.

We now have rough position information for all objects in the image. The next task is to make full use of the information in the gray image to enhance the position data to sub-pixel accuracy.

2.2 Step II: Fine Scan

2.2.1 General Fitting

The central feature of the algorithm is the adapting of parameters of given models for the intensity image to the real measured data, thus gaining the highest possible accuracy. Having the rough position approximations we have to choose the right model out of all possible ones described in chapter 3.2. The situation is illustrated in Figure 4. The adapting itself is done using the MatLab build-in functions, which implement a least-square solver. One additional advantage of the method is, that an error estimation is fairly easy, as described in chapter 3.3.2.

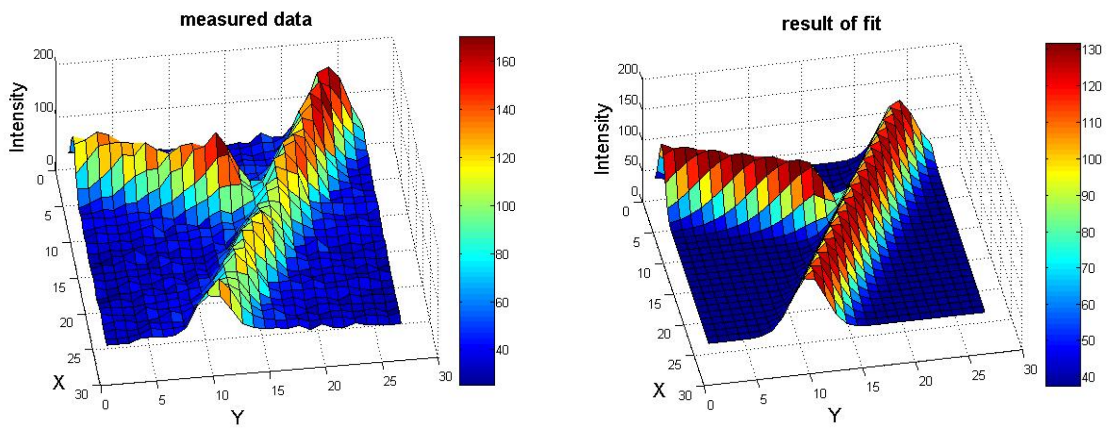

Fig. 4: *The original data (left) and the approximation using a theoretical model (right).*

The provided models are tuned to represent the natural shapes best, but because of interference there occur problems, if objects are very close to each other. The solution is to take a sum of several models, each describing one of the objects. It is assumed, that the intensity can be simply added up to get the combined model. Finding the parts, where this has to be done is task of the

2.2.2 Cluster Analysis

Each object covers a certain area of interest, where its influence to the image is considerable high. These areas are called regions and are rectangles of size `fit_size` having the center at the point estimation for the object. The constant `fit_size` is fixed to 2.5 times the width estimate.

The first step in the fine scanning process is to find those areas (called clusters), where objects come close to each other or even intersect[2], such that their regions overlap. Another requirement for this analysis is, that it locates distinct[3] clusters. The closer two objects are to each other, the more important it is that these parts are included into a region. On the other hand, it is necessary that regions don't get too large, because otherwise the model approximation does not hold anymore and additionally the computation would take longer. To find an optimal solution to these competing requirements the following approach is used:

1. Find all regions where two objects come close and store the coordinates in a list of dimension N. Objects are considered to be close if there distance is below four times the given length scale.

2. Order this list by the distance of the two objects, where small distances are at the top.

3. Calculated the pairwise overlap of all region combination possibilities and store it in a N-by-N matrix. The overlap of two rectangles is defined as the area of the intersection divided by the area of the row region.

[2] We already resolved crossing objects in the previous step, but there might be the possibility, that two regions are so close together, that their intensity signals overlap, although the thresholding separated them into distinct parts. It is therefore more robust to totally neglect the thresholding and determine close lying objects independently.

[3] This is necessary, because ideally no data points should be used for fitting twice to avoid any weighting.

4. Get the first region of the list and take it as a first approximation for a cluster.
5. Get the region with the highest overlap to that cluster and take the smallest rectangle containing both the cluster and the region as the new cluster.
6. Calculate the area of the new cluster.
7. As long as the area is below the given threshold, continue with adding more regions at step 5 otherwise go on with the next step. The area threshold is fixed at 15 times the square of the given length scale.
8. Delete all regions of the list, which are covered by the cluster. This step is necessary to make sure no regions are processed twice and the clusters are not badly influenced by the presence of the other uncovered objects nearby. The algorithm makes sure, that the disregarded regions are not that important with respect to the defined requirements, because their overlap is lower than the included regions.
9. Store the final cluster in the cluster list.
10. Continue with a completely new cluster at step 4, if the region list is not empty.

After that procedure, the region list is empty and regions, where objects come close are stored in the cluster list, which will be processed in the next step.

2.2.3 Fitting Clusters

Now that we found all difficult parts we may analyze them in detail. It is important to pick the right model for the objects in each cluster and fit it in a single process to include interference phenomena. As pointed out before, we use a sum of our basic models, where the initial estimates are easily taken form the rough scan process. These parameters are enhanced using the least-square fitting described in more detail in chapter 3.3.

2.2.4 Post Processing Clusters

After fitting the cluster with the initial estimates we remove false points using these catagories:

- Distance between two beads is smaller than the sum their radial fitting errors
- Distance between two beads is smaller than the sum their gaussian widths

2.2.5 Scanning Remaining Points

Now that all complicated parts have been handled, only the easy ones remain, such that the program just runs through all points on all objects and fits the estimated position (and orientation) with a suitable model. It is important that not all intermediate points on an elongated object are taken, because otherwise some data points are used several times and are therefore weighted heavier. On the other side it is very important to include the end points in a fit to have precise knowledge of the length of the object.

2.3 Step III: Further Calculations

As a last step, some further calculations are performed for each object to ease analysis. They may be separated into two categories: *Interpolation*, to get information on intermediate points which were not fitted directly and *Averaging* to get mean properties of the whole object.

2.3.1 Interpolation between points

The goal for this part is, to have a list of dense points along the middle line of the elongated object with the usual parameters width, height and background level. For that purpose both the position and the orientation of the located points of the fitting step are taken into consideration. The interpolation is done using piecewise third-order polynomials between two consecutive points, which are well-defined by the position and the orientation of the two points. The length of the segment is the curve length, which than may be subdivided in equidistant parts of an approximated length of one pixel. The spline interpolation is described in detail in section 3.5.1.

Using all the segments, one can create graphs of the properties width, height and background level depending on the length on the elongated object. The values are known at some points (the interpolation points of the previous step) and may be easily estimated in-between using the MatLab build-in cubic interpolation. For the height profile a different method is used, where the original gray image is taken into consideration: The height is taken directly from the image at the exact calculated position using a 2-dimensional linear interpolation approach.

The final data is saved in a separate substructure named `data`. Because of the nature of interpolation, it is hard to calculate any errors.

2.3.2 Averaging of properties

For some applications it is necessary to have properties of the whole object – for example a position of the center or an average width of the object.

All these values are extremely easy to calculate if the object is point-like, because in this case the values of the fit of that point are exactly the ones we are looking for. For an elongated object the goal may be accomplished by taking the weighted average of the respective properties of the fitted points. An elongated object is a list of points with the necessary properties connected by a spline of known length. The approach for calculating averages is now, that each point gives a contribution to the value directly proportional to the length of the adjoining splines. In this way both the values (position, height, width, background level) and their errors may be easily calculated.

The above calculation for positions leads to an averaged point that is somehow like the *center of mass* of the object. Due to the calculation this point does not have to lie on the object itself. To make up for this, a second definition of the center of the object is used: The point on the contour of the object at exactly half the total length. As stated in the previous section, there are no meaningful easy error values for the spline interpolation and therefore it is not possible to calculate errors for the second type of center.

3 Theory

3.1 Thresholding

Calculating the black and white binary image is the crucial step of the whole algorithm. The program supports two different methods:

fixed Taking a threshold level, which has to be provided by the calling program via the parameter `threshold`, the gray image is processed in the following way: All pixels having values above the given threshold are set to one, whereas the other ones are set to zero. To remove artifacts, a closing and opening of the binary image [6] is done. Using this way bright regions are assumed to be the important objects.

adaptive This method takes no input and tries to estimate all parameters itself. The basic feature detection is done using edge detection [7] taking the SOBEL-method[4]. The returned image is already binary, where the boundaries of objects are set to one. The task is now to fill the contours in the right way to get a nice binary image. After basic artifact removal using the `bridge` and `fill` method of `bwmorph()` [6], larger holes are filled, if there mean intensity value is above a threshold, which is determined automatically by adding the mean value and the standard deviation of the whole gray image. For better illustration this part of the source code is shown in the Appendix A.1. Essentially, this method also emphasizes the bright regions.

The user may choose which method suits his data best, as the first one allows more detailed adaption – especially for bad signal to noise ratio, but the second one is easier to use and may yield better results for inhomogeneously illuminated images. Another advantage of the adaptive method is the possibility to find even very dark objects, which otherwise will be removed. The speed difference of both methods may be neglected, but the first method is naturally faster.

To increase the quality of the created binary image, additional filters may be applied for the fixed threshold method in certain situations. It is the users task to choose the filter which suites the problem best:

none No filter will be applied and the above description is fully correct.

average before The gray image will be processed with an 3-by-3 average filter, thus removing unwanted extremely bright or dark pixels, which most likely will be artifacts. This filter is best, when there is a bad signal-to-noise ratio, but the objects are larger than a pixel.

smooth after The image will be processed after thresholding to remove some unwanted artifacts. An morphological closing followed by an opening will be done, thus removing very small objects and smoothing the boundaries of larger ones. This filter is best used for tracking large elongated objects. It is necessary to smooth the boundary, because otherwise the thinning procedure used for finding elongated objects will produce small additional objects perpendicular to the right one.

With respect to computational effort, the first method is naturally the fasted, but these filters anyways don't take up much time in comparison to the fitting done afterwards.

[4] Using an approximation to the derivatives in both x and y direction, one can find the parts of the image, where intensity changes most, leading to an estimate for boundaries in general.

3.2 Used models

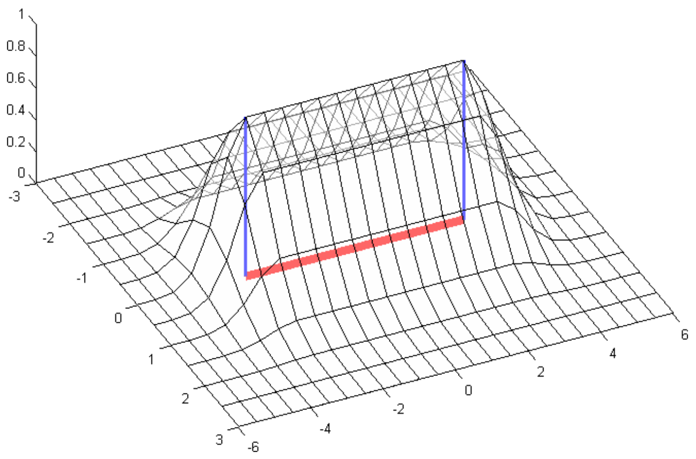

Fig. 5: *Microtubule (red) with convolution (black). The blue vertical lines mark the endpoints.*

The theoretical models have been chosen with the idea of convolution in mind. The objects under the microscope are so small, that their shape may not be seen directly, but is blurred with the *point spread function* (PSF) of the optical setup. The assumption of the algorithm is, that this PSF is a Gaussian.

To cover the two different types of objects that should be detected, there are several models that describe the intensity I of the image based on position variables (x, y) and a small set of parameters. The different parameters are optimized using a least square method to minimize the difference between the model and the measured data. The aim is to estimate the original position of the fluorescent objects. This method is also known as deconvolution. An important application of this is the determination of the end points of a microtubule as seen in Figure 5. The black grid is the intensity distribution (without noise) seen in the image. By using the right model the blue lines correspond to the end points of the underlying red object, which has been convoluted by the PSF. The position of the object itself should ideally correspond to the position of the ridge of the picture. The width of that function should be given by half of the wavelength of the emitted light, because the objects are small compared to the resolution of the microscope.

Considering these facts there are only a few kind of shapes that might occur in a rectangular part of the image, which are described in detail in the following sections:

3.2.1 Symmetric 2D-Gaussian

This is the standard two-dimensional Gaussian with center $\hat{x}$ and $\hat{y}$, width σ and an amplitude h:

$$I(x,y) = \frac{h}{2\pi\sigma^2} \cdot \exp\left[-\frac{(x-\hat{x})^2 + (y-\hat{y})^2}{2\sigma^2}\right] =: A \cdot \exp\left[-\frac{(x-\hat{x})^2 + (y-\hat{y})^2}{B}\right] \tag{1}$$

To speed up calculation a bit, some transformed variables A and B have been introduced for the height and width, respectively.

This model is used for a point-like object, e.g. some quantum dots or magnetic beads. A more advanced version may be constructed if one does not require a symmetric width:

3.2.2 Stretched 2D-Gaussian

This version of a 2D-Gaussian is the more general case, where the width is not the same in every direction, but can be described by two widths σ_x and σ_y and a correlation factor ρ forming an ellipse:

$$\begin{aligned} I(x,y) &= \frac{h}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \cdot \exp\left[-\frac{1}{2(1-\rho^2)}\left(\left(\frac{x-\hat{x}}{\sigma_x}\right)^2 - 2\rho\,\frac{x-\hat{x}}{\sigma_x}\,\frac{y-\hat{y}}{\sigma_y} + \left(\frac{y-\hat{y}}{\sigma_y}\right)^2\right)\right] \\ &=: A \cdot \exp\left[-\left(\frac{x-\hat{x}}{B_x}\right)^2 - \left(\frac{y-\hat{y}}{B_y}\right)^2 + C(x-\hat{x})(y-\hat{y})\right] \end{aligned} \tag{2}$$

where once again some new variables with the following correlation have been introduced in the last line:

$$\begin{aligned} A &:= \frac{h}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \\ B_x &:= \sigma_x\sqrt{2(1-\rho^2)} \\ B_y &:= \sigma_y\sqrt{2(1-\rho^2)} \\ C &:= \frac{\rho}{\sigma_x\sigma_y(1-\rho^2)} \end{aligned}$$

The last three lines have to be transformed, such that the inverse relation can be used in the program:

$$\begin{aligned} \rho &= \frac{CB_xB_y}{2} \\ \sigma_x &= \frac{B_x}{\sqrt{2(1-\rho^2)}} \\ \sigma_y &= \frac{B_y}{\sqrt{2(1-\rho^2)}} \end{aligned}$$

This model might be used for point-like objects, if the illumination is not perfect and the dots in the image are not circular.

A complete different type of objects are the elongated ones (e.g. microtubules), which in general are described by a section of a curve. To ease calculation we only consider linear parts such that the three following possibilities might occur in the rectangular area used for fitting:

3.2.3 Middle part of elongated object

The middle part of elongated objects are modeled using a Gaussian wall, which is described in the following way:

$$I(d) = \frac{h}{\sigma\sqrt{2\pi}} \cdot \exp\left[-\frac{d^2}{2\sigma^2}\right] =: A \cdot \exp\left[-\frac{d^2}{B}\right] \tag{3}$$

where d is the distance of the point (x, y) to the ridge. In the easiest model, the ridge is a straight line $\vec{a}(t)$ described by a reference point $\vec{p} \equiv (\hat{x}, \hat{y})$ and a slope expressed by the angle φ enclosed by the x-axis and the line itself:

$$\vec{a}(t) = \vec{p} + t \begin{pmatrix} \cos\varphi \\ -\sin\varphi \end{pmatrix}$$

To use the model $I(d)$ for our approach we need the intensity I in dependence of the arbitrary point $\vec{q} = (x, y)$. To calculate needed properties it is necessary to define the ray $\vec{b}(s)$ starting at $\vec{q}$ forming a right angle with the ridge:

$$\vec{b}(s) = \vec{q} + s \begin{pmatrix} -\sin\varphi \\ -\cos\varphi \end{pmatrix}$$

Using these two straight lines, the crossing point and especially the two distances t_r and s_r from this point to the reference point $\vec{p}$ and the arbitrary point $\vec{q}$, respectively, may be calculated:

$$\begin{aligned} t_r(x, y) &= (x - \hat{x}) \sin\varphi + (y - \hat{y}) \cos\varphi & (4) \\ s_r(x, y) &= (x - \hat{x}) \cos\varphi - (y - \hat{y}) \sin\varphi & (5) \end{aligned}$$

Using this results, two models are constructed for the Gaussian wall:

$$\begin{aligned} d_1(x, y) &= s_r(x, y) & (6) \\ d_2(x, y) &= s_r(x, y) + c \cdot t_r^2(x, y) & (7) \end{aligned}$$

where the first one describes a linear ridge and the second a quadratic curved one[5]. For the second model, c is a curvature parameter, which is zero for a straight line. Using the later approach a generalization to higher order polynomials is unproblematic.

3.2.4 End of elongated object

End points of elongated objects are a little bit more complicated to describe because they are the superposition of half of a (linear) Gaussian wall for the elongated part and half of a Gaussian hill for the actual end. It is sufficient to take the coordinates $(\hat{x}, \hat{y})$ of the end point and an orientation angle φ to specify both parts in space. It is important that the superposition is done smoothly by using a antialaised half-plane to cut out the important parts from a full wall and hill, respectively.

3.2.5 Small elongated object

The final and most complicated model includes two end points and is therefore the superposition of two half Gaussian hills and a wall in-between. The spacial description can be done using the coordinates of the two end points. This model is used for very short objects, which can be notifiable different from elliptic point objects.

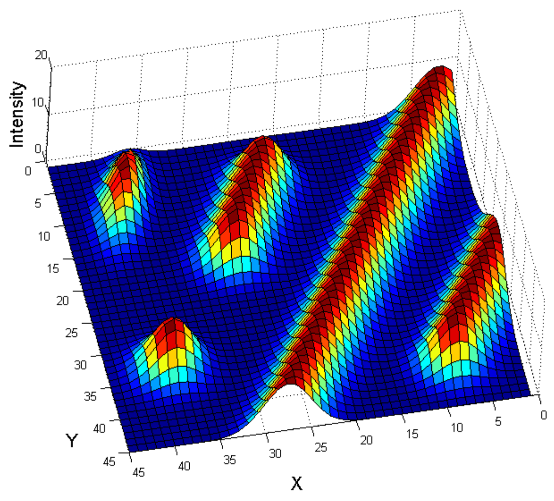

Fig. 6: *Image of all defined models: elliptic, circular, small elongated object, middle part and end part of elongated object(f.l.t.r.)*

3.2.6 Summary

To illustrate the defined shapes more easily, we made an artificial sample image seen in Figure 6, where one can see all models in counterclockwise order starting at the top left corner. The shown objects are well distinguished, but overlapping is not a problem as demonstrated in Figure 4.

Using one model for point-like and the above three ones for elongated objects we can describe and fit any possible image by cutting out rectangular parts and taking a superposition of the right models. The size of the rectangle is crucial. If it is too large the approximation for the ridge is bad and additionally the fitting process takes a lot longer. On the other hand, if the area is too small, the fitting has not enough data points to get an estimation of the parameters with a small error.

Technically, all the models are implemented as different classes of the same structure, thus making it easy to include additional models in the program code without changing much. Each of the classes takes care of specific operations, such as guessing parameters, the evaluation step, transforming results, etc.

3.3 Fitting approach

The most important part of the whole algorithm is the fitting, which is used to adapt the parameters of the given models to the acquired image. Luckily, there are very good and robust algorithms available. In fact, we use the MatLab `lsqnonlin()` function, which is an implementation of a general least square solver. The used models and the way to determine the guesses for the parameters are described in the section above.

Additionally, the fitting is computational intense, thus taking up most of the time of the scanning. To speed things up, the needed derivatives of the models are calculated exactly using

[5] It is not really quadratic, because the exact shape naturally has to be a Gaussian integrated along the ridge line. Assuming the curvature is not too high, the model nonetheless yields good results.

the reverse mode technique of automatic differentiation (AD). To ease programming a little bit, the source code has been transformed into FORTRAN code to be processed with the AD-tool TAPENADE [8]. The result has been transferred back to MatLab and enhanced manually a little bit.[6] The advantage of using exact derivations over finite difference approaches, which is used by default, is that the program is faster and yields more accurate results with the setback of a little bit more complex code. All derivatives have been tested by comparing them to the finite differences.

3.3.1 Used coordinate system

It might be helpful to specify the used coordinate system exactly[7], and how coordinates are transformed during the processing. Transforming is necessary, because the image has to be cropped to confine the fitting region. Throughout the whole program, integer values refer to the center of a pixel, where the top left pixel has the coordinates $(1, 1)$. The confined fitting region is determined in the following way:

1. Guesses $\vec{g}$ are calculated using centroids in the thresholded image as described in chapter 2.1.1.
2. The centroid, which should be the center of the fitting region, is the average of all guesses.
3. As regions must be on the pixel lattice, an integer center point has to be found. This is achieved using the MatLab `imcrop()` function, which crops the image and returns the corresponding (integer valued) rectangle.

3.3.2 Error estimate

The errors of parameters determined in the fitting process are calculated the same way as Origin[9] does it in its widely used non-linear fitting tool. One has to be careful about the results, because they are only applicable if the model fits the data. A totally wrong model may never give any meaningful results. The approach used for the error of a certain approximation is very general:

Assuming a problem has N data points depending on p parameters, the Jacobian matrix $\mathcal{F}_{ij} = \partial N_i/\partial p_j$ returned by the MatLab `lsqnonlin()` function (or calculated directly in our case), is a N-by-p matrix. The expression for the error of the i-th parameter is than [10]

$$\sigma_i = \sqrt{\mathcal{C}_{ii}\chi_r^2} \qquad i = 1, \ldots, p \qquad \text{with} \qquad \mathcal{C} = (\mathcal{F}^\mathsf{T} \cdot \mathcal{F})^{-1} \tag{8}$$

where C is the p-by-p variance-covariance matrix and $\chi_r^2 = \frac{\chi^2}{N-p}$ is the reduced chi-squared value computed using the normal χ^2 of the least-square-method divided by the number of data points N minus the number of parameters p. The χ^2 is the sum of the squared deviations between the model and the original data.

[6] Enhancing includes simplifying and reducing expressions, which is extremely hard to do automatically. Additionally, we don't need the full result with additionally possibilities.

[7] In general, there exist two different types for bitmap pictures, where integer values either denote a corner or the center of a pixel. Additionally, in contrast to ordinary applications, in computer graphics the origin lies in the top left corner.

The error of the length of an elongated object is mainly influenced by the error in the position of the endpoints. The errors in x- and y-direction are independent and therefore define an ellipse. The diameter of the ellipse in direction of the orientation of the endpoint can be calculated and used to estimate the length error. Doing this procedure at both sides and adding up half of the diameters yields the error estimate:

$$\Delta l = \sum_{i=\mathrm{s,e}} \sqrt{(\Delta x_i \cdot \cos\varphi_i)^2 + (\Delta y_i \cdot \sin\varphi_i)^2}$$

where i denotes the start (s) and end (e) point of the elongated object. Especially for long objects, the length error is usually underestimated, although the overall accuracy is a lot better than measuring the length by eye.

3.3.3 Propagation of uncertainty

All error values are stored in a new MatLab class `double_error` which contains double precision values with assigned errors. The class supports most basic numeric operations and takes care of the propagation of errors. Assuming we have values x_j with uncertainties δx_j and a function $y = f(x_1, ..., x_n)$ used for calculation of a new value y, the task is to calculate the uncertainty δy. This is usually done using a total differential method, applicable if the errors are uncorrelated[8]:

$$\delta y = \sqrt{\sum_{j=1}^{n} \left(\frac{\partial f}{\partial x_j}\delta x_j\right)^2} \quad \Rightarrow \quad \delta y \lessapprox \sum_{j=1}^{n} \left|\frac{\partial f}{\partial x_j}\right| \delta x_j \tag{9}$$

Normally the terms should be added up in a squared way – as seen in the left equation – which is very expensive computational wise. The method (9) use by the program is much faster and produces only slightly larger errors.

The error calculation is implemented using a special class `double_error` that takes care of the above formula for all necessary operations by overloading the respective methods. The class stores both the values x and their errors δx.T

3.3.4 Validation

The goodness of the fit is estimated using the *coefficient of determination* (CoD) R^2 [11]:

$$R^2 = 1 - \frac{\text{sum of squared errors}}{\text{total sum of squares}} = \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} \tag{10}$$

where the y_i are the N data points, $\hat{y}_i$ are the fitted points and $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$ is the mean value of the data.

Additional verification is done using the actual values of the fitted parameters. For example, the height of objects is compared to the standard deviation of the background, such that very dark objects, which are accidentally fitted due to the nature of the thresholding algorithm, are disregarded.

[8] This is not true for all cases in the program and should be noticed in the analysis.

3.4 Units of Returned Values

The calculated parameters are the position, orientation, width, height and background level at each point. The dimensions of each value are the following:

positions are given in pixel times the scaling set by the external parameter `scaling`. The axis are the natural ones of the image where the origin is in the upper left corner.

orientations are given in counterclockwise orientated angles out of the interval [0,2*pi), where the zero direction points to the right

widths are FWHM (Full Width at Half Maximum) values multiplied by the scaling, which are connected to the GAUSSIAN width σ: $\text{FWHM} = 2\sqrt{2\ln 2} \cdot \sigma \approx 2.35 \cdot \sigma$

heights are in the units of the gray scale of the image. They are the difference between the maximum point and the background level.

background level is the calculated background level in the units of the gray scale image

3.5 Interpolation

3.5.1 Spline interpolation

Between two consecutive fitted points on the elongated object, a spline interpolation is used to gain detailed information about the curve. The used initial data are the two positions of the points together with their orientations. The possible fitted curvature for middle points is neglected for the interpolation, because the error of this value is large and the influence on the splines is quite high. The interpolation algorithm consequently works in the following way:

1. Rotate, translate and scale the points and orientations in space, such that point A is in the origin of an ordinary x-y-coordinate system and point B lies at (1,0) on the positive x-axis.
2. Use a cubic polynomial with four unknown parameters and determine these, such that the function passes through both points with the right slope determined by the orientation. The result is a list of four parameters determining the special cubic function.
3. Transform the parameters such that the rotation, translation and scaling of step 1 is reversed. The result of the back-transformation are two list with four parameters for the x and y direction, respectively.

The result may be interpreted as an parametric equation $t \mapsto (x(t), y(t))$ for both axis. Each component is a cubic function, where the parameter is in the range $t \in [0,1]$. The length between the two points may than be easily calculated by numeric integration:

$$L = \int_0^1 \sqrt{\dot{x}^2(t) + \dot{y}^2(t)}\, dt$$

4 Usage of algorithm

The user does not need to call any of the supplied routines himself, but should rather use the FOTS interface to access them. Nevertheless here is a brief summery of how to use the methods directly from within MatLab. Additionally, some extra information is given to ease analysis and tracking down problems.

4.1 ScanImage.m

The function `ScanImage()` does all necessary tasks to scan one image and returns the retrieved data in an array of `structs`. It requires two parameters, where the first one is the gray image itself and the second one is a `struct` contains advanced settings to influence the procedure. The returned array has one entry for each object found, containing the following fields:

time the creation time of the frame, that has been passed to the algorithm as the parameter `creation_time`

center_x the x-coordinate of the center

center_y the y-coordinate of the center

com_x the x-coordinate of the center of mass and its error

com_y the y-coordinate of the center of mass and its error

orientation the orientation of the object, which is the angle of line between the two end points and the x-axis

height the averaged height of the object with an estimated error

width the averaged width of the object with an estimated error

length the length of the object and an estimated error. This value is zero if it is an point-like object.

p an array of length N containing the data of the fitted points on the object. N is the total number of points, where each one has the following fields attached:

- **x** two position coordinates
- **o** an optional[9] orientation value
- **w** values for the width of the object. If the general bead model is used, this field contains three values.
- **b** the background level at this point
- **h** the height of the point

All these values are of the type `double_error` thus containing error estimations as well.

data an array containing computed information on the object. The interpolated points are approximately one pixel apart. Each point has the following fields attached:

[9] The orientation is only used for elongated objects.

- **x** the x coordinates
- **y** the y coordinates
- **l** the distance on the elongated object ranging from zero to the full length
- **w** the width as a Gaussian sigma. It is determined by a cubic interpolation of the fitted values of the points in the `p` array.
- **b** the background level at this point determined via cubic interpolation.
- **h** the height of the point. This is the value of the image at this specific point.

These values have no errors attached!

The units of the returned values are stated in section 3.4.

4.2 Input Parameters – Settings

All parameters are passed as a `struct` to `ScanImg()` as the second argument after the actual image. The `struct` has to contain the following fields:

fwhm_estimate is an estimation of the object FWHM-width in pixel. It is usually determined by the wave-length of the laser and the resolution of the camera. This information is necessary for the program to set a length-scale on which it should look for any objects.

Additionally, the following fields may be included, to influence the behavior of the program:

area_threshold is used for distinguishing between point-like and elongated objects. Objects with lower area as given in area_threshold are treated as simple ones. (default: 30)

bead_model is stating which model should be used to fit point like objects. The two possible values are `circle` and `ellipse` corresponding to the two models mentioned in section 3.2 (default: `ellipse`)

binary_image_processing sets a method for some special processing of the binary image. Several types are available, which are described in more detail in Section 3.1. (default: `none`):

Method	Effect
`none`	no processing
`average`	does an averaging over a 3x3-box of the gray image before applying the threshold
`smooth`	does a smoothing of the borders of the binary image after applying the threshold

border_margin is the amount of pixels an object has to be away from the border, until it is taken into account. (default: 0)

bw_regions: A boolean image of the same size as the image, containing the regions, which should be processed. The given data is multiplied with the binary image of Step I, thus ruling out regions, which are marked black (zero) in `bw_regions`. (default: completely white image)

creation_time is a value for the time, where the image has been created. It is stored for every found object for later analysis. It does not influence the algorithm at all. (default: NaN)

display: An integer flag determining the level of output:

Value	Effect
0	no output (except for fatal errors)
1	output to MatLab Command Window
2	graphical output for debugging
3	both graphical and text output

If there exists a variable `logfile` in the global scope of MatLab, the program assumes its a file handle and tries to write its text output to this file instead of the console. (default: 0)

find_beads is a boolean value denoting, if the algorithm should look for point like object. (default: true)

find_molecules is a boolean value denoting, if the algorithm should scan for molecules. (default: true)

height_threshold is used for a post selection process, where the height of objects is compared to the standard deviation of the background. Objects with a ration of height to standard deviation lower than the height_threshold are disregarded. (default: 5)

max_beads_per_region is a parameter used for restricting the maximum number of beads, that may be considered in one region. (default: Inf)

MaxIter: Maximum number of iterations for each fitting call.

MaxFunEvals: Maximum number of function evaluations for each fitting call.

min_cod: A threshold for the Coefficient of Determination defined in equation (10). Fitting results, which produce lower CoD-Values, are disregarded. (default: 0.5)

ridge_model is the model used for the middle parts of the elongated objects. Supported options are `linear` and `quadratic`. See chapter 3.2.3 for further details. (default: `linear`)

scale: A factor denoting a factor used for scaling all length information by simple multiplication. (default: 1)

short_object_threshold denotes the maximum length in pixels of a short elongated object, which is fitted in one step. Larger objects are processed in small steps, where each time only a small area is regarded. (default: 15)

threshold: A global threshold value to determine the black and white image. It is not recommend to use this value, because the internal procedure has an adaptive behavior, thus leading to a better result in general.

TolFun: Termination tolerance on the function value for a fitting call.

TolX: Termination tolerance on the parameters for a fitting call.

4.3 Error events

The algorithm uses the global variable `error_events` to trace common problems and events. The struct consists of several integer fields, each counting the number of occurrences of one kind of problem. The following list explains the different types:

touching_border: A region is too close to the border and therefore ignored. This may be controlled by the parameter `border_margin`

cluster_cod_low: The CoD of a fit of a cluster is below the threshold set by `min_cod`.

endpoint_cod_low: The CoD of a fit of an endpoint is below the threshold set by `min_cod`.

middlepoint_cod_low: The CoD of a fit of a middlepoint is below the threshold set by `min_cod`.

bead_cod_low: The CoD of a fit of a point-like object is below the threshold set by `min_cod`.

MT_cod_low: The CoD of a fit of a small elongated object is below the threshold set by `min_cod`.

degenerated_MT: The fit of a small elongated object yielded a point object (both end points ended up in the same place).

empty_object: An empty object occurred, where all points have been neglected due to other reasons.

point_not_fitted: A critical error, where the algorithm forgot to consider a point.

found_wrong_type: Point like objects have been found, although only elongated ones were requested or vice versa.

object_too_dark: The height of the object, which is equivalent to its intesity, was below the threshold set by `height_threshold`.

fit_hit_bounds: The parameters of the fittings exceeded the bounds normally indicating the data does not match the chosen model.

fit_impossible: The fitting algorithm could not handle the problem. This is most likely due to a problem with the input data, which might not be suited for this algorithm.

area_too_small: Regions have been disregarded, because their area has been below the given `area_threshold` and only elongated objects should be tracked.

Some of these are no serious problems and might occur due to the nature of the algorithm. They may be used to properly tune the input parameters, e.g. if a lot `cod_low` errors occur, one might consider lowering the CoD threshold.

5 Warnings and known problems

- **Assumptions**
  - There are no other features in the image, apart from point-like and elongated objects and some noise, which is ignored in the way described above.
  - The shape of the objects is Gaussian – this is well fulfilled, if the objects size is below the wave-length of the emitted light.
  - The intensity is simply summed up at crossings (which might not be true for certain imaging methods)
- **Given coordinate axis**

  The algorithm is a little bit biased towards the given axis, because we use rectangular regions for fitting.
- **Global variables**

  For speed reasons the algorithm uses the following names as global variables: `xg`, `yg`, `fit_pic`, `pic`, `bw`, `error_events`. It can not be guaranteed that these variables have the same values as before after the script ran. This is just important, if the algorithm is used by another MatLab program.

  Additionally, the global variable `logfile` is checked. If it is a MatLab file handle, the text output is redirected to that file. Otherwise it is set to an empty array.

6 Further improvements and outlook

- Better criteria on when to keep objects or not. This would be helpful to disregard objects that were found accidentally.
- Use time dimension to determine where possible objects may lie and if one big object might be a cluster of two smaller ones.
- Analyze the whole movie to find frames where the algorithm found two consecutive objects, although there has been one long object in reality, which might be recognized in other frames.

References

[1] G. T. S. Brian C Carter and S. P. Gross, "Tracking single particles: a user-friendly quantitative evaluation," *Physical Biology*, vol. 2, no. 1, pp. 60–72, 2005. [Online]. Available: http://stacks.iop.org/1478-3975/2/60

[2] M. K. Cheezum, W. F. Walker, and W. H. Guilford, "Quantitative comparison of algorithms for tracking single fluorescent particles," *Biophys. J.*, vol. 81, no. 4, pp. 2378–2388, October 2001. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/11566807

[3] F. Pampaloni, G. Lattanzi, A. Jonas, T. Surrey, E. Frey, and E.-L. Florin, "Thermal fluctuations of grafted microtubules provide evidence of a length-dependent persistence length," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 103, no. 27, pp. 10 248–10 253, July 2006. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/16801537

[4] D. M. W. Steven S. Work, "Computer-assisted tracking of actin filament motility," *Anal Biochem*, vol. 202, pp. 275–285, 1992. [Online]. Available: http://www.cytoskeletons.com/showabstract.php?pmid=1519753

[5] C. P. Brangwynne, G. H. Koenderink, E. Barry, Z. Dogic, F. C. Mackintosh, and D. A. Weitz, "Bending dynamics of fluctuating biopolymers probed by automated high-resolution filament tracking," *Biophys. J.*, vol. 93, no. 1, pp. 346–359, July 2007. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/17416612

[6] *Description of the MatLab `bwmorph` function*, The MathWorks, Inc., [accessed 14-November-2007]. [Online]. Available: http://www.mathworks.com/access/helpdesk/help/toolbox/images/bwmorph.html

[7] *Description of the MatLab `edge` function*, The MathWorks, Inc., [accessed 14-November-2007]. [Online]. Available: http://www.mathworks.com/access/helpdesk/help/toolbox/images/edge.html

[8] *TAPENADE - Automatic Differentiation*, INRIA Sophia-Antipolis, [accessed 14-November-2007]. [Online]. Available: http://www-sop.inria.fr/tropics/tapenade.html

[9] *Origin Fitting Concept*, OriginLab, [accessed 14-November-2007]. [Online]. Available: http://www.originlab.com/www/helponline/origin/The_Fitting_Session_Concept.htm

[10] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipies in C*, 2nd ed. Cambridge University Press, 2002.

[11] Wikipedia, "Coefficient of determination — wikipedia, the free encyclopedia," 2007, [accessed 14-November-2007]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Coefficient_of_determination&oldid=171210976

A Parts of the source code

The complete source code has been developed using MatLab Version 7.5 (R2007b) using the `Curve Fitting`, `Image Processing` and `Optimization` Toolboxes.

A.1 Converting gray to binary images

The gray image is assumed to be given in `input`. The binary image will be returned in `output`. User-made settings are given in `threshold`, which is a number for the chosen constant threshold or unset, if the automatic threshold detection should be used. Additionally, there is the value `param.binary_image_processing` to set a filter which should be applied. Both these options are described in section 3.1.

```
if ~isempty(threshold) % apply simple threshold

  % average gray image, if requested
  if strcmp( params.binary_image_processing , 'average' )
    input = conv2( input , fspecial( 'average', 3 ), 'same' );
  end

  % appply threshold
  output = ( input > threshold );

  % smooth binary image, if requested
  if strcmp( params.binary_image_processing , 'smooth' )
    output = bwmorph( output , 'close' );
    output = bwmorph( output , 'open' );
  end

else % apply automatic threshold

  automatic_threshold = mean2( img ) + std2( img );

  % find edges of objects
  output = edge( input , 'sobel', [], 'both', 'nothinning' );

  % close small gaps
  output = bwmorph( output , 'bridge' );
  % fill one-pixel holes
  output = bwmorph( output , 'fill' );

  % fill bigger holes in each object
  l = bwlabel( imcomplement( output ), 4 );
  l_props = regionprops( l , 'Area', 'Image', 'BoundingBox', 'PixelIdxList' );
  f = find( [ l_props.Area ] < 50 );
  for i = f
    region_gray = imcrop( double( img ), l_props(i).BoundingBox - [ 0 0 1 1 ] ) ...
                  .* l_props(i).Image;
    if mean2( region_gray ) > automatic_threshold
      output( l_props(i).PixelIdxList ) = 1;
    end
  end

  % multiply with low threshold image to rule out very dark areas
  output = output .* ( img > automatic_threshold );

end
```