Optimizing Pipeline for 3D Reconstruction of Protein Aggregates

Catarina Gonçalves *ist1102654*, Guilherme Costa-Ferreira *ist1103394*, and João Freitas *ist1103218*

***Abstract*—Throughout different realities of the medical domain, it's essential to study cellular structures due to the mechanism's complexity and the biological processes. To do so, microscope observations are critical, and one of the most popular methods of structure elucidation in cell physiology is fluorescence microscopy. Given 28 histological slices from one cell containing two protein aggregates obtained by this technique, we applied fundamental steps from imaging processing. Moreover, this project has the purpose of reaching into a 3D reconstruction of the protein aggregates, which underlie the pathogenesis of neurodegenerative disorders, as well as computing the volume of these structures. Starting by removing characteristic noise from each slice through the Gaussian filter (after choosing it, compared to the median filter), we proceed to the segmentation and binarization phase, using the Search2Segment algorithm, a method developed by our group, which integrates the well-established Otsu method. During each stage, different parameters were tested and compared, aiming to preserve edges that shape aggregate form. With the masks applied, we assembled and aligned them to reconstruct the protein aggregates and calculate their volume. Based on a voxel dimension, we got to the total volume equal to 170.40 μm^3. All code and data in the GitHub: https://github.com/GuilhermeCosta-Ferreira/3DReconstruction**

***Index Terms*—Denoising, Segmentation, 3D Reconstruction, Registration, Volume.**

I. Introduction and motivation

IMAGING methods for segmentation, 3D reconstruction or any methodology that allows us to study, from images, the cellular world are powerful tools highly demanded in the medical realm. These approaches allow us to diagnose certain conditions, analyse the effect of a drug or even study the biological processes that take place in our body.

The reconstruction of a surface from a set of planar cross-sections S (slices) has been extensively studied in the past decades. These slices, once segmented in an initial step, form a series of closed two-dimensional shapes that delineate the boundary between the inside and outside of an object on each layer. A typical reconstruction method generates a surface that encloses a volume, which interpolates these contours and is consistent with the predefined inside/outside distinctions.

Previously established methods in this area generally presume access to full cross-sections with complete, accurately segmented contour information—meaning each cross-section is expected to provide a complete set of closed shapes. However, in real-world applications, the original images often contain noise, leading to areas of uncertainty where segmentation may not be dependable.

In this report, the problem to solve starts with 28 images registered by a fluorescence microscope, an important optical microscope in biological studies, monitors cell physiology and diagnostic imaging due to its molecular specificity and super-resolution (in some situations) in the range of 15 to 20 nm [1]. Furthermore, its microscope design has the specific goal of increasing image contrast and spatial resolution [2]. Looking at the methods and analysis that we will describe later in this report, it's important to pay attention directly to the added noise with complex sources by the capture process and also the structures that will be reconstructed.

As a matter of fact, two main types of noise are described in these situations: detector noise and shot (or dark) noise [3]. The first one is caused by the randomness of the photon-counting process (based on the concept of light as a stream of discrete photons) in the camera or photodetectors. This leads to the signal-dependent Poisson component that when the amount of signal increases, the amount of dark noise also increases - although at a slower rate. The second type of noise is related to the intrinsic heat and electronic fluctuations of the acquisition devices (electronic measurement), leading to a signal-independent Gaussian component. As described, this noise will not depend on the signal, but it is intrinsically determined by the type of detectors used.

From a mathematical perspective, we can describe these two types of noise and relate them with the measured signal:

$$y_i = x_i + n = x_i + [g(x_i) + p(x_i)] \tag{1}$$

where y_i is the measured experimental signal (composed of different pixel values), x_i is the ground truth image, n represents the noise, which can be decomposed on its two previously described components, Gaussian ($g(x_i)$) and Poisson ($p(x_i)$) noise [4].

During this project, we focused on three-dimensionally reconstructing protein aggregates. Before describing our approach, let's explain the biological importance of this formation: although it can be functional, as observed in physiological mechanisms, in some cases, it can be caused by disrupted protein folding, mutations or decreased protein stability, being linked to certain diseases including Alzheimer and Parkinson [5].

Based on the literature, we developed this work with an iterative approach, validating the state-of-art, but always adapting each method to our context. As illustrated in the figure 1.

This project started (after introducing all images as input in MatLab) with image restoration, through filtering to remove as much noise as possible. Following this step, we segmented two aggregates of proteins and partitioned these from other constituent parts of the cell and background. After that, it was applied masks to these 2D results to obtain a 3D configuration

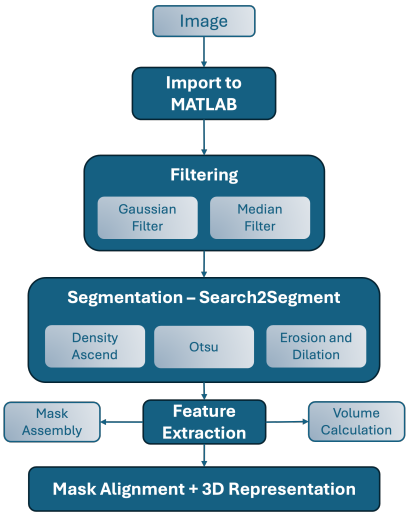


Fig. 1: Diagram of the used pipeline

and then, we aligned those to approximate the real representation. Finally, we compute the volume of the aggregates.

II. State of the art

Initial studies on reconstruction from contours (e.g., Batnitzky et al. [1981], Fix and Ladner [1998]and Fuchs et al. [1977]), focused on interpolating between parallel cross-sections of an object. A subset of this research (including Choi and Park [1994], Christiansen and Sederberg [1978] and Ekoule et al. [1991]), specifically addressed the simpler case of reconstructing from slices each containing a single contour. However, these early methods often struggled with complex shapes, leading to issues like branching anomalies or the generation of self-intersecting geometries that were considered unsatisfactory.

Research by Bajaj et al. [1996]; Barequet and Sharir [1996], Boissonnat [1988], and others from the mid-1990s to early 2000s explored the reconstruction of a single object from parallel slices, without limiting the shapes, geometries, or arrangements of the contours within these slices. Their work primarily concentrated on developing segments of the object's surface between adjacent slices and then assembling these segments to form the complete boundary surface [6].

More recent methods address the reconstruction of shapes from unorganized (non-parallel) and irregularly spaced cross-sections [7].

Most algorithms for reconstructing from cross-sections utilize a geometric approach, where they match vertices and edges of polygonal contours to create triangulated surfaces. This often involves projecting contours onto one another to solve vertex correspondence in 2D. The research by Barequet and Vaxman [2009] and Liu et al. [2008] involves organizing cross-section planes, then projecting the contours from each cell's boundary onto the straight skeleton or medial axis within the cell. Following this, they construct "walls" connecting the original contours with their projections to form sections of the boundary surface.

While these methods are generally fast and work well for many cases, they tend to produce surfaces with rough triangulations that require subsequent smoothing. Moreover, because the calculations involve nonrational coordinates, which can be inaccurate for designs supposed to have precise measurements, they lack numerical robustness. This issue is critical since minor errors in constructing the medial structure can result in incorrect surfaces that intersect themselves or have gaps [6].

The intersection of 3D reconstruction with deep learning is a rapidly growing area, with deep learning models providing detailed insights into shapes and structures based on large 3D object datasets [7]. These models produce excellent results at the expense of training time and complexity. Due to this, the general path of improving these strategies involves easier hyper-parameterization selection methodologies [8].

III. Problem description

We consider a set $P = \{P_1, \ldots, P_k\}$ consisting of 2D planes situated within the three-dimensional space $\mathbb{R}^3$. Each plane P_i encompasses a distinct set of oriented, non-overlapping contours $C_i = [c_{i,1}, \ldots, c_{i,l_i}]$. These contours effectively divide each plane into defined regions that are either "inside" or "outside" a certain unknown domain Ω located in $\mathbb{R}^3$, which is delineated by the boundary $\partial\Omega$. Fundamentally, we start with 28 matrices like 2048x2048, where each position is filled with its respective intensity value and we pretend to conclude this work with a tensor 2048x2048x28, where only aggregate voxels are non-zero intensity components.

IV. Methods and Results

This section includes the topics "(Mathematical) Problem Formulation", "Methods and Algorithms" and "Validation and Testing".

A. Filtering

As described earlier, the first step was based on choosing the most appropriate filter and applying it. Contextualizing our search it is reported throughout various sources that for a specific image with Gaussian [9] and Poisson [10] noise, the Gaussian filter gives reliable and good results.

Another type of filter applied to processing medical images is the median filter, which is proven to have a better performance at grey images corrupted with Poisson noise [11], being capable of removing outliers without reducing image quality [9] [12].

Hence, we tested filtering the set of 28 images with median and Gaussian distribution approaches. Moreover, multiple articles use the mean filter and compare it to others, mainly concluding that it is not [13] [9] [14] well effective at reducing noise without influencing the edge contour and features distinction.

It's important to mention that for fluorescence microscope images, more sophisticated filtering techniques are developed and applied (as Non-local means and wavelet transforms [3]) leading to really good results, but in our case, we chose simpler and straightforward approach by using median and Gaussian filters.

The median filter was applied using the *medfilt2* function. It is a non-linear spatial filter applied with a 2-D mask that replaces each pixel's value with the median of the intensity values in its neighbourhood. The median is calculated by first sorting all the pixel values from the window into ascending numerical order, and then replacing the pixel being considered with the middle (median) pixel value. It preserves the edges while removing noise and smoothing the image.

This filtering does not introduce new pixel values, as it reuses existing ones from the window. The median is less sensitive to extreme values (outliers) and, therefore is better able to remove these outliers without reducing the sharpness of the image [15].

Although capable of significantly reducing noise with even a small 3x3 filter size, larger filters may lead to loss of detail and image structure [16].

Then, this filter can be formulated as:

$$y(n) = \text{median}([x(n_0 - p...n_0 + p]) \tag{2}$$

where 2p+1 corresponds to the window size and n_0 the current pixel.

Median filter can blur noise, highlighting areas of interest such as bright circles.

It is observed that with the [3,3] window median filter, the smoothing is relatively light. In this case, the median is calculated from 9 values, which allows to remove small noise and fine details, but preserves the overall structure. By increasing the window to [10,10], the median is calculated from 100 values, resulting in much stronger smoothing. Larger noises and loudness variations are eliminated, but finer details may be lost and be significantly smoothed. This window introduces more blur into the image (figure 2).

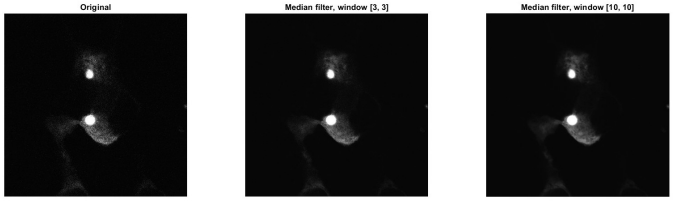


Fig. 2: Slice 11: original, with median filter window [3x3], with median filter window [10x10]

Focusing now on the Gaussian filter, we can describe it as a linear low-pass filter. It is considered an efficient low-pass filter, compared, for example, with a simple box filter. This can be explained by the impact caused in the frequency domain due to time domain convolution. First, note that this convolution (described in Expression 3) corresponds to the product of the same two functions in the frequency domain. Besides that, contrary to the box filter, where its Fourier Transform gives a sinc function, the Gaussian filter, through the Fourier Transform, maintains its Gaussian shape, suppressing high frequencies (associated with noise) and allowing only low frequencies to pass.

Moreover, it uses a Gaussian kernel. In other words, each weight is defined according to a Gaussian distribution, decreasing its value while increasing the spatial distance to the centre position. We proceed to a weighted average of the neighbouring points related to a single pixel, substituting then this value. At this point, it's clear to think about the smoothing effect created (blurred images), although, provoking loss of edge sharpness and reduced details.

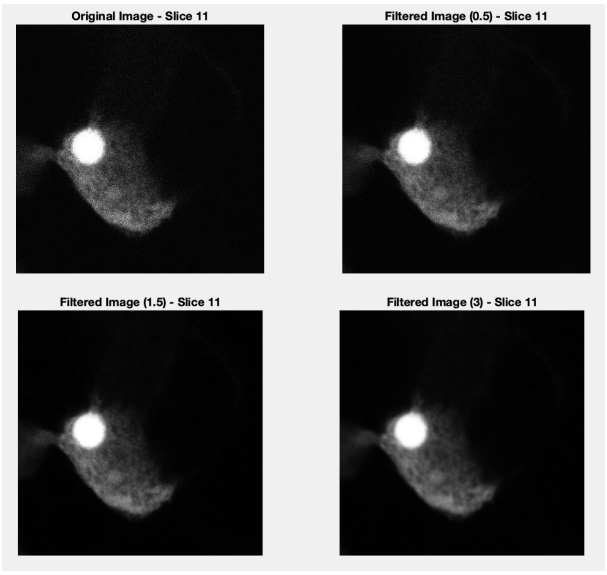


Fig. 3: Different values of σ applied in Gaussian filter to slice 11 - zoomed in to the lower aggregate

Regarding computational costs, these mathematical operations (multiplication and addition) strongly depend on kernel size and the number of pixels in the image.

Before presenting our results, we need to highlight the standard deviation σ influence in Gaussian filter behaviour (the distribution is assumed to have a mean of 0). Described by the expression:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3}$$

the smoothing effect gets stronger if we increase σ value - this is coherent with the 1D representation: the higher the σ gets, the flatter the Gaussian curve become. Consequently, the expression for the filtered image is the following:

$$y(n) = \sum_{r \in S} G_\sigma(r) x(n - r) \tag{4}$$

where r and n are coordinate vectors and S represents the coordinate space around the current pixel.

Looking at Figure 3, it's now evident the impact of σ when its value is altered, proves what was described earlier. When this parameter is higher, the image loses its original granularity, also having a higher blurred effect.

Another essential parameter that has to be considered is the size of the filter. Briefly, we can also mention a large window size, which results in a prominent Gaussian blur. Additionally, it is related to σ. According to the literature, it is recommended that the standard deviation be no larger than the window size, but more specifically, that filter size should be three times the standard deviation, in a way that neighbouring pixels that are at a distance more than 3σ away do not contribute to the current's pixel new value.

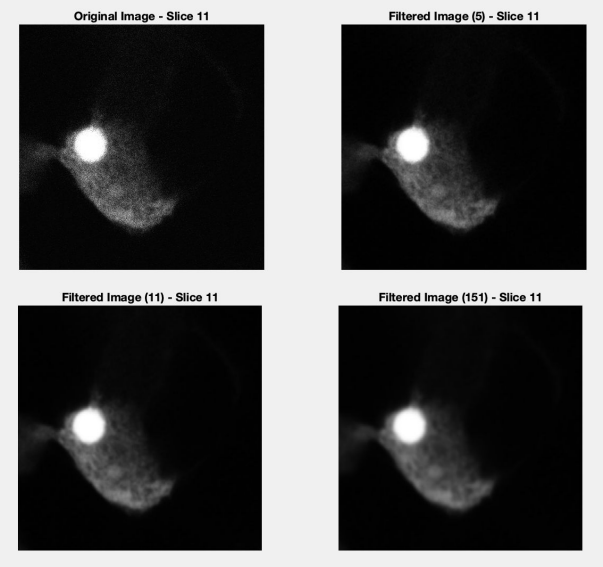


Fig. 4: Different kernel sizes applied in Gaussian filter to slice 11 - zoomed in to the lower aggregate

By default, *imgaussfilt* function in MATLAB uses a kernel of size: $(2 \cdot \text{ceil}(2\sigma) + 1) \times (2 \cdot \text{ceil}(2\sigma) + 1)$. To verify the standard ratio between size and standard deviation, we tested three different kernel sizes with the same σ, equal to 1.5 (due to the well-established results, according to our goal).

Noting that the first image of the second row corresponds to the standard ratio - σ equals 1.5 and kernel size equals 11 (3σ for each side, plus the central pixel), it's evident the smoothing effect that represents well this type of filter. The last image shows when a excessive kernel size is selected, resulting as well in excessive blurred pixels, losing the wanted contour definition of each aggregate.

To proceed into the next phase of our process, we chose the Gaussian filter, justified by the common use reported in literature, specific to fluorescent microscopy. Regarding to this filter, we maintained the default kernel size (associated with almost the same results as the standard ratio) and used the 1.5 σ value, preserving the edges of the structures and not introducing unwanted high blur to the image. On the other hand, we didn't consider the median filter appropriate to our case, corrupted by Poisson-Gaussian noise and not Salt and Pepper (more indicated to the median filter).

B. Segmentation

The next step in the pipeline (figure 1) is segmentation, for which the literature is vast, with many approaches taking the spotlight in different occasions and scenarios. This step receives the filtered image and should return binarized images with only the protein aggregates. The solution found can be categorized into two distinct categories: the traditional approach and the deep-learning approach.

To classify these different methods three important variables are analysed: the segmentation accuracy, precision and recall (that is, metrics of how effective the methods are in object detection) and the segmentation computation time.

The more traditional approaches include non-adaptive algorithms which can not only identify but also segment images in one go. In this realm, we have methods like otsu, k-means, mean shift or Chan-Vese that hold dice similarity coefficient (DSC) of 94%-95% [17]. However, the computational times vary greatly with the Otsu method being the quickest per image: 1.08 sec/image and the Chan-Vese the slowest: 330 sec/image [17].

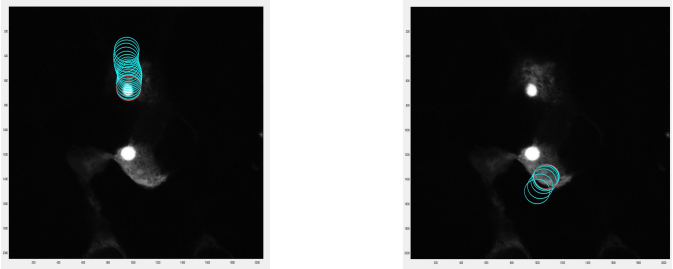
Traditional segmentation methods like Otsu are very good and very quick in identifying the desired object but fail in being sufficient for the 3D reconstruction, commonly post-processing is needed. For our work, we built a three-step segmentation algorithm that we called: Search2Segment.

Search2Segment Algorithm
(1) Search and Mask the aggregates with the Density Ascend (DA) Algorithm;
(2) Binnarize with the Otsu Method;
(3) Erosion and Dilation;

The Otsu method by itself might hurt the sensibility for 3D reconstruction but allows for a quick solution. The question arose in our team: How can we improve the Otsu performance? The main problem with the histogram approach was the presence of other clumps of intensity in the image. Even between aggregates, one could have a spike in different intensities in images where they did not saturate. If we could isolate one aggregate at a time, then Otso could work with a piece of simpler information and be more effective. A new approach was born: we added a step before the Otsu, based on the mean shift algorithm: the Density Ascend Algorithm.

Density Ascend Algorithm
(1) Place a starting circle near the bright object you want to segment;
(2) New center calculated with the equation 5;
(3) Move the circle to the new centre (if applicable reduce its size);
(4) Iterate until the resting position has been found or the number of iterations has reached the imposed limit;
(5) Start in the new next image by placing the starting circle in the final coordinates from the previous image;
(6) Iterate over all images and objects;
(7) Mask the image with the final circle position and radius;

Let's, for instance, try and isolate the lower aggregate. If we create a mask, let's say a circle, that would "walk" into more dense image regions, then, after some iterations, it would naturally arrive at the aggregate position. Once there if we reduce all the information to 0 outside said mask, Otsu could work with a simpler histogram and produce better results. The circle's walk would work this way: It would calculate the mean intensity inside its radius and start the next iteration in the newly calculated centre. This new position would be calculated following the equation 5.



(a) Aggregate found (b) Aggregate not found

Fig. 5: Algorithm test-run on image 11

$$\begin{bmatrix} \sum_{i=0}^{N} x_i \cdot p_i \\ \sum_{i=0}^{N} y_i \cdot p_i \end{bmatrix} \cdot \left(\sum_{i=0}^{N} p_i \right)^{-1} \tag{5}$$

Where x_i are the x-coordinates inside the mask and p_i are the corresponding intensity values (between 0 and 1), the analogous logic for the y_i, but for the y-coordinates. The search would end when the previous circle centre equals the new centre. We then run just the search algorithm with different radius and starting positions:

After some tests we understood better the new challenges this new approach had. For instance, the radius size was defined as 100 pixels, since the biggest aggregate was a few pixels smaller. Also, we stuck with a circle mask (the images in figure 5 are distorted in the x-direction) since the aggregates' size was practically round. The more challenging parameter was the starting point. If we started in a different position it could get stuck in an "intensity plateau". The quick fix was to try, by hand, to start in a more or less close-to-the-aggregates position. The other, more efficient fix, was more clever. We should remember that we are using images that are not independent from each other, that is, the following image is just a higher transversal section of the same object. With this in mind, we made the starting circle start from the previous final centre. This greatly fixed the plateau situations in the middle images.

Another problem that was reduced with this solution was that the algorithm was computationally intense. If previously a worse start position could mean 100 iterations in each image for each aggregate, the new update made the next images only need some 5-15 iterations. Another update to battle the high computing times was the addition of a sensibility margin. Previously, we had 0 tolerance, that is, the search would only finish if the centre was in the exact same position as the previous centre. This new iteration added some tolerance, it could stop if the centre was 1-2 pixels near the previous one.

Another solution to the plateau issue was the addition of a starting radius and an ending radius. With a bigger starting radius, the Density Ascend Algorithm had better "vision", being able to move quickly into high-density zones. While moving, the radius would reduce until it reached the ending radius, that is, as we got closer to the objective, the algorithm would "focus" and make more sensible adjustments. Not only that, but the smaller the radius the better the Otsu performed. This circle reduction is evidenced in the figure 6.

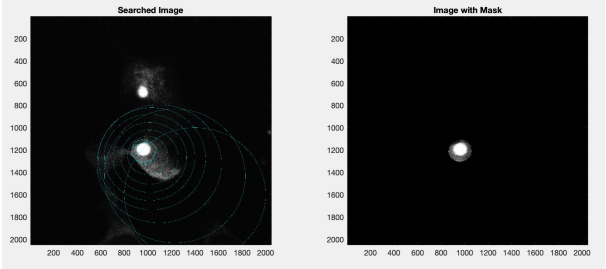


Fig. 6: Image 11 search algorithm with reducing circles applied

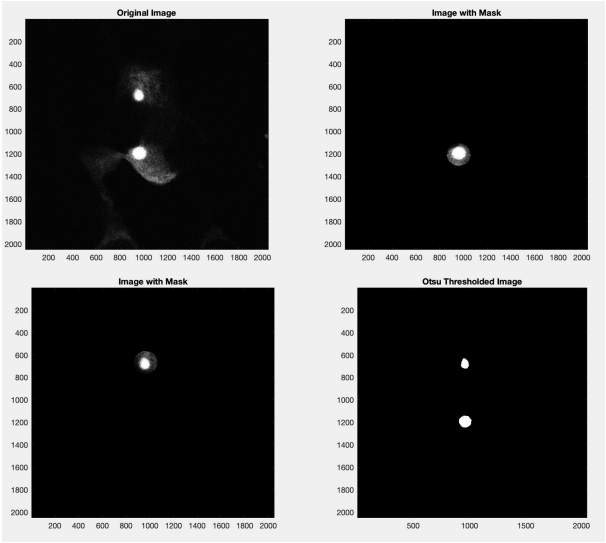


Fig. 7: Algorithm's steps visualization the higher aggregate in image 11

If the user made a terrible starting choice or swapped the x and y coordinates by mistake, the code could run hundreds of iterations before finding a resting position. We added an early stop mechanism that would forcefully stop the run if some iteration number was reached.

Once the the mask was finished, we applied the Otsu method avoiding big tinkering actions required by a brute stand-alone Otsu method. We found that running the code for each aggregate separately and then adding the results provided better results despite the slightly longer execution time. Ultimately the sensibility was improved as evidenced by the figure 7.

To understand the different efficiency and advantages that each alteration provided our approach we run three different versions: (1) without any optimization (brute DA algorithm); (2) all optimizations without circle reduction (optimized DA algorithm); (3) all optimization plus circle reduction (Dynamic DA algorithm). With these three versions, we then run them for 3 different scenarios: (1) Terrible starting circle; (2) Medium starting circle; (3) Great starting circle;

With five iterations per condition we obtained the results presented in the figure 8.

All the data was added in the appendix, including the code specifications used to generate said results.

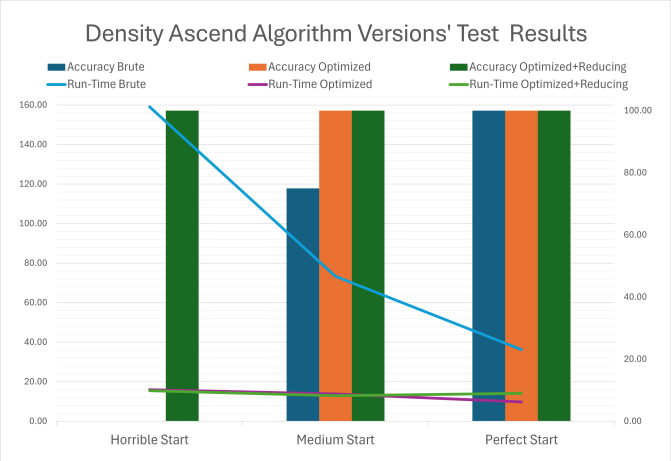


Fig. 8: Test results for DA algorithm versions' accuracy and run-time

As we can see, the addition of optimizations greatly reduced the algorithm run-time from 36.95 ± 2.27 sec to 9.76 ± 0.45 sec in perfect conditions. However, as conditions deteriorated, the algorithms' accuracy started reducing. This is where the reducing circles update shined with the dynamic DA algorithm improving from 0% accuracy in the worst conditions to 100%, without sacrificing much time.

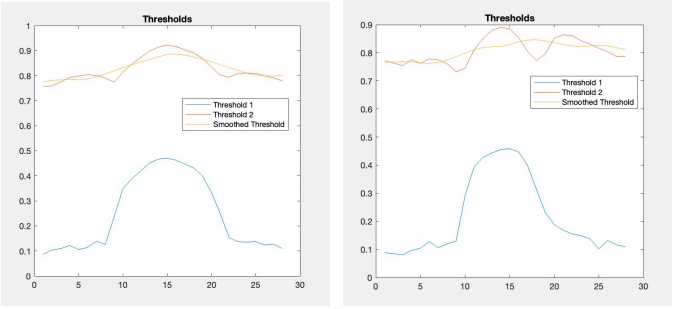
Otsu Algorithm and Binarization
(1) Calculate the multiple thresholds by maximizing the between-class variance (equation 6);
(2) Smooth the threshold function with a moving average algorithm;
(3) Binarize each object's image with the corresponding smoothed threshold value;
(4) Re-add the binarized object images from the same plane;

The DA algorithm was needed due to the nature of the Otsu algorithm. Otsu takes the image's pixel intensity histogram, searches for spikes and returns the threshold that, when binarized, isolates the desired spike. When in the presence of multiple spikes we can opt for the Matlab function `multitresh` which returns all the found thresholds. In our case, we found better results with the second threshold.

To calculate, for instance, one threshold let's take the desired image to binarize and normalize its histogram. This way the histogram works as a probability function. Let's define an arbitrary k threshold that divides the image into C_0 and C_1. The Otsu algorithm will try to maximise the between-class variance $\sigma_B^2(k)$ for each k [18]. Where this variance was obtained in formula 6

$$\sigma_B^2(k) = \frac{[\mu_T \cdot \omega_0 - \mu_0 \cdot \omega_0]^2}{\omega_0 \cdot \omega_1} \tag{6}$$

Where the μ_T is the histogram's mean and ω_0 and ω_1 are the probability of class C_0 and C_1, respectively. To search for the second threshold, one can follow an analogous formulation



(a) Thresholds for the lower aggregate (b) Thresholds for the higher aggregate

Fig. 9: Otsu Threshold Visualization

but for three classes (C_0, C_1 and C_2). Once k is found, we binarize, that is every intensity value inferior to k becomes 0, and the rest 1s.

Since, in each image, the second threshold might change a bit, we used a smoothed threshold function to convey the aggregates when binarizing adequately. This function can be visualized in figure 9.

Erosion&Dilation Algorithm
(1) Remove the pixels which do not fit the segment element, eliminating noise;
(2) Fill pixels which were hit by the segment element, recovering signal lost to erosion;

Because this step might create some noise we added some post-processing by applying some erosion and dilation. Erosion is a method that, with a given segment element (SE), will only preserve pixels which have a similar neighbourhood to the SE. We made the SE a very small disk shrinking only lost pixels from the binarization step.

In other words, the function `imerode` from Matlab will take the SE and try and fit in the image (f). If it fits, then it won't erode (equation 7) [19].

$$f(x,y) \ominus se = \begin{cases} 1 & \text{if } se \text{ fits } f(x,y) \\ 0 & \text{c.c} \end{cases} \tag{7}$$

Where $f(x,y) \ominus se$ is the erosion of image $f(x,y)$ by the segment element se.

Since erosion might remove some data from the signal we applied dilation. This method tries to fill the image with missing pixels. In practical terms, it smoothed our final result. In analogy, instead of trying to fit the se to $f(x,y)$ we will try and "hit" [19].

$$f(x,y) \oplus se = \begin{cases} 1 & \text{if } se \text{ hits } f(x,y) \\ 0 & \text{c.c} \end{cases} \tag{8}$$

It is important to clarify that, our method Search2Segment, is a segmentation method optimized for low-count 3D reconstruction. If we tried and used it for cell counting, or multiple cell 3D reconstruction, the algorithm run-time would make this approach impractical. It is optimally used for a few objects and will produce highly detailed results.

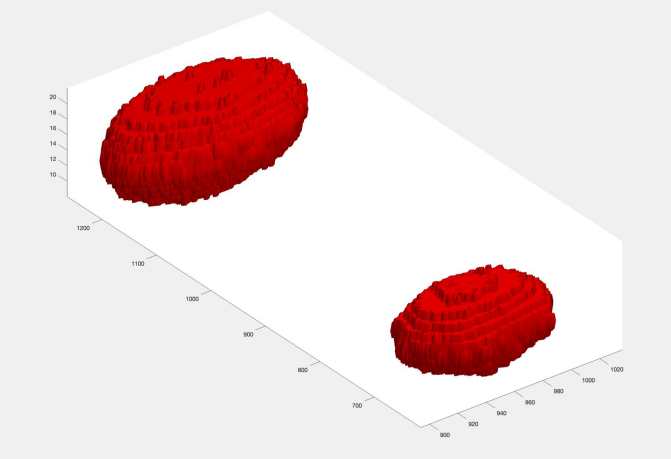


Fig. 10: 3D reconstruction from the segmented image data.

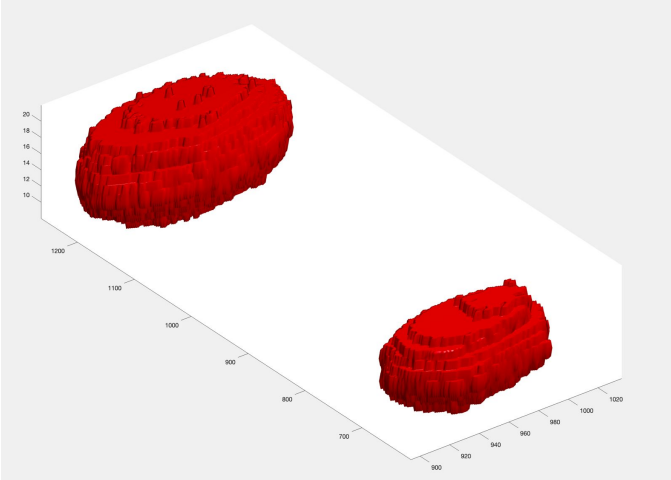


Fig. 11: 3D reconstruction of aligned segmented images.

C. Assembling the masks to build a 3D representation of the aggregates

After completing the filtering and segmentation processes, the goal was to compile the individual masks to create a 3D representation of the aggregates. A tensor initialized to zero was prepared, with dimensions dim*dim*l, where dim=2048 and l=28 (number of segmented cross-sectional images). The tensor was loaded with the images obtained after filtering and segmentation.

To obtain the image in 3 dimensions, we used the *isosurface* function – that computes and draws a surface by connecting points of a constant value within a volume of space, effectively delineating the boundary of the object of interest in 3 dimensions [20]. It was used a threshold of 0.5 distinguishing between the segmented objects and the background.

The *patch* function provides the flexibility to manipulate how the constructed 3D object is presented, allowing for adjustments in visual properties such as the coloration of the surfaces and the edges [21]. The visualization parameters are adjusted—tight axis limits, a uniform aspect ratio, and strategic lighting conditions—to optimize the 3D representation's clarity and detail. Figure 10 represents the 3D reconstruction from the segmented image data.

D. Applying registration methods to align the masks of the cross-sections

To ensure that the information is properly integrated, all contributing images must be accurately aligned. This process is called image registration, which aims to spatially align two images that might have been captured under differing conditions, such as variations in orientation, spatial resolution, or imaging modality. These discrepancies can manifest as shifts, rotations, scaling, or distortion in the overlapped images. The primary goal of image registration is to minimize these differences to achieve a seamless integration.

In the process of registration, it is crucial to designate one image as the target (or fixed) image and the other as the transformable (or moving) image. Typically, the image with a higher spatial resolution is chosen as the fixed image to serve as a reference point. In this case, was chosen slice 15 as the fixed image.

The MATLAB functions *imregconfig* and *imregister* were used. *imregconfig* is used to set up the registration configuration, chosen as 'monomodal' since it was assumed that all images have consistent brightness and contrast. The *imregister* function is used to perform rigid registration, which involves finding the optimal rotation and translation that aligns the images with the reference image without altering its shape (no scaling or shearing).

This method effectively standardizes the orientation and position of segmented images, ensuring that all images are oriented in the same way relative to the chosen reference. Figure 11 represents the 3D reconstruction of the aligned segmented images.

Analysis of figure 11 shows that, after alignment, one aggregate remained unchanged, indicating that its initial orientation closely matched the reference frame. The upper layer of the second aggregate appears to have shifted slightly, suggesting a misalignment in the original scans that was corrected during the alignment process to achieve consistency across the dataset.

E. Volume

To compute the volume of each aggregate, we used the scale provided to discover the length and the width of each pixel - $64.285\,\mu$m - knowing that our image is a 2048×2048 pixels matrix. This lead to dimensions equal to 3.139×10^{-2} micrometres. We then review the literature to discover the thickness of each slice. Usually, for fluorescent microscopy (confocal microscopy, for example), thickness varies between 0.5 and $1.5\,\mu$m. So, we considered the value of $1\,\mu$m to the last dimension of each voxel, enabling to calculate the volume.

Outline the number of white voxels (three-dimensional pixels) of each of the clusters, the volume of each aggregate was obtained by multiplying the number of voxels by the volume of each voxel.

Table I presents the values obtained for the volume of each of the aggregates.

	Lower Aggregate	Higher Aggregate	Total
Volume per voxel (μm^3)	9.85×10^{-4}		
Number of voxels	114631	58310	172941
Total volume (μm^3)	112.95	57.45	170.40

TABLE I: Volumes obtained from the aggregates.

V. Conclusions

The problem of reconstruction from parallel cross-sections has been studied extensively and there exists a significant body of literature on this topic.

In this project, to perform 3D reconstruction from 28 2D fluorescence microscopy images, an iterative approach combining advanced image processing techniques for the segmentation, 3D reconstruction, and alignment of cellular aggregates is presented.

Filtration of the 28 fluorescence microscopy images was performed with a Gaussian filter, which is reported to give reliable and good results for a specific image with Poisson noise, preserving the edges and not introducing unwanted high blur to the image.

The subsequent segmentation phase receives the filtered image and returns binarized images with only the protein aggregates. We built a three-step segmentation algorithm, the Search2Segment algorithm, demonstrating the utility of combining traditional thresholding techniques with innovative search algorithms to isolate regions of interest effectively. It includes: search and mask the aggregates with the Density Ascend (DA) Algorithm; Binnarize with the Otsu Method and Erosion and Dilation.

Assembling the masks obtained in the segmentation operation was done to build a 3D representation of the aggregates and then registration methods were applied to align the masks of the cross-sections, where small misalignments were observed. By employing rigid registration techniques, we were able to correct these misalignments, ensuring that each slice was accurately positioned relative to a fixed reference, thus facilitating a coherent reconstruction of the cellular aggregates in three dimensions.

We compute the volume of these aggregates, having obtained the 112.95 μm^3 and 57.45 μm^3 values for each one.

Quantifying aggregate volumes offers invaluable insights for applications from drug efficacy to pathological studies. Future work is needed to refine these techniques further and explore their applicability to a broader range of biological phenomena.

References

[1] S. C. M. Reinhardt, L. A. Masullo, I. Baudrexel, P. R. Steen, R. Kowalewski, A. S. Eklund, S. Strauss, E. M. Unterauer, T. Schlichthaerle, M. T. Strauss, C. Klein, and R. Jungmann, "Ångström-resolution fluorescence microscopy," *Nature*, vol. 617, no. 7962, p. 711–716, May 2023. [Online]. Available: https://www.nature.com/articles/s41586-023-05925-9

[2] M. J. Sanderson, I. Smith, I. Parker, and M. D. Bootman, "Fluorescence microscopy," *Cold Spring Harbor Protocols*, vol. 2014, no. 10, p. pdb.top071795–pdb.top071795, Oct 2014. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4711767/

[3] R. F. Laine, G. Jacquemet, and A. Krull, "Imaging in focus: An introduction to denoising bioimages in the era of deep learning," *The International Journal of Biochemistry & Cell Biology*, vol. 140, p. 106077–106077, Nov 2021.

[4] G. Chen, J. Wang, H. Wang, J. Wen, Y. Gao, and Y. Xu, "Fluorescence microscopy images denoising via deep convolutional sparse coding," *Signal Processing: Image Communication*, vol. 117, p. 117003–117003, Sep 2023.

[5] N. Majid and R. H. Khan, "Protein aggregation: Consequences, mechanism, characterization and inhibitory strategies," *International Journal of Biological Macromolecules*, vol. 242, p. 125123–125123, Jul 2023.

[6] A. Bermano, A. Vaxman, and C. Gotsman, "Online reconstruction of 3D objects from arbitrary cross-sections," *ACM Trans. Graph.*, vol. 30, no. 5, pp. 1–11, Oct. 2011.

[7] https://bit.ly/3IZ2QkQ, [Accessed 29-03-2024].

[8] A. Wang, Q. Zhang, Y. Han, S. Megason, S. Hormoz, K. R. Mosaliganti, J. C. K. Lam, and V. O. K. Li, "A novel deep learning-based 3d cell segmentation framework for future image-based disease detection," *Scientific Reports*, vol. 12, no. 1, Jan 2022.

[9] S. Das, J. Saikia, S. Das, and N. Goni, "A comparative study of different noise filtering techniques in digital images," *International Journal of Engineering Research and General Science*, vol. 3, no. 5, 2015. [Online]. Available: https://pnrsolution.org/Datacenter/Vol3/Issue5/25.pdf

[10] E. G. Onyedinma and I. E. Onyenwe, "Image restoration: A comparative analysis of image de noising using different spatial filtering techniques," *International Journal of Latest Technology in Engineering, Management & Applied Science*, vol. XII, no. XI, p. 55–63, 2023. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/2401/2401.09460.pdf

[11] N. Kumar and M. Nachamai, "Noise removal and filtering techniques used in medical images," *Oriental journal of computer science and technology*, vol. 10, no. 1, p. 103–113, Mar 2017.

[12] R. Shankar, "Noise reduction in image using matlab," *www.academia.edu*, 2020. [Online]. Available: https://bit.ly/3PG62Wl

[13] M. F. Nasution, Sriani, and I. F. Nasution, "Noise reduction using gaussian, mean, and median filter to improve ultrasonography image quality," *INFOKUM*, vol. 10, no. 1, p. 505–513, Dec 2021. [Online]. Available: https://infor.seaninstitute.org/index.php/infokum/article/view/327

[14] L. Shi, Y. Chen, W. Yuan, L. Zhang, B. Yang, H. Shu, L. Luo, and J.-L. Coatrieux, "Comparative analysis of median and average filters in impulse noise suppression," *Fluctuation and Noise Letters*, vol. 14, no. 01, p. 1550002, Dec 2014.

[15] "Using Median Filtering :: Analyzing and Enhancing Images (Image Processing Toolbox User's Guide) — matlab.izmiran.ru," http://matlab.izmiran.ru/help/toolbox/images/enhan23b.html, [Accessed 31-03-2024].

[16] jun94, "[CV] 2. Gaussian and Median Filter, Separable 2D filter — medium.com," https://medium.com/jun94-devpblog/cv-2-gaussian-and-median-filter-separable-2d-filter-2d11ee022c66, [Accessed 31-03-2024].

[17] K. Y. Win, S. Choomchuay, K. Hamamoto, and M. Raveesunthornkiat, "Comparative study on automated cell nuclei segmentation methods for cytology pleural effusion images," *Journal of Healthcare Engineering*, vol. 2018, p. 1–14, Sep 2018.

[18] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, p. 62–66, Jan 1979. [Online]. Available: https://ieeexplore.ieee.org/document/4310076

[19] O. Marques, "Morphological image processing," *John Wiley & Sons, Inc. eBooks*, p. 299–334, Aug 2011.

[20] "Extract isosurface data from volume data - MATLAB isosurface — mathworks.com," https://www.mathworks.com/help/matlab/ref/isosurface.html, [Accessed 31-03-2024].

[21] "Create patches of colored polygons - MATLAB patch — mathworks.com," https://www.mathworks.com/help/matlab/ref/patch.html, [Accessed 31-03-2024].

Appendix

Density Ascend Algorithm Versions's Specks

The following section defines the parameters that run each version used for testing: (1) Brute DA Algorithm; (2) Optimized DA Algorithm; (3) Dynamic DA Algorithm.

Brute Density Ascend Algorithm

- Early Stop: 200 iterations
- Radius: 100 pixels
- Tolerance: 0 pixels
- Stored Start: False
- Reducing Circles: False

Optimized Density Ascend Algorithm

- Early Stop: 200 iterations
- Radius: 100 pixels
- Tolerance: 1 pixel
- Stored Start: True
- Reducing Circles: False

Dynamic Density Ascend Algorithm

- Early Stop: 200 iterations
- Starting Radius: 600 pixels
- Ending Radius: 100 pixels
- Reducing Iterations: 20 iterations
- Tolerance: 1 pixel
- Stored Start: True
- Reducing Circles: True

These models were tested in different scenarios. The final version of the Density Ascend Algorithm is the Dynamic Version, as this one provided better results.

Testing Scenarios

To evaluate each model we subjected each one to three different scenarios where the starting position changed quality: (1) Horrible Start; (2) Medium Start; (3) Perfect Start. Each matrix represents two starting circles. The first row is the x and y coordinates for the circle that will search for the lower aggregate. The second row is for the higher aggregate.

Horrible Start

$$\begin{bmatrix} 1000 & 2000 \\ 900 & 0 \end{bmatrix}$$

Medium Start

$$\begin{bmatrix} 1000 & 1000 \\ 900 & 400 \end{bmatrix}$$

Perfect Start

$$\begin{bmatrix} 1000 & 1200 \\ 900 & 600 \end{bmatrix}$$

Obtained Data

To evaluate the models in these different scenarios we run the code 5 times for each condition, registering the following variables:

Observed Variables

- Search Time: Time it took the algorithm to search all 28 images in the dataset;
- Nº Early Stops: Number of times the search was terminated not because it found a resting position, but because it reached a limited number of iterations;
- Nº Deviation to Other Aggregates: Number of times the circle moved and stopped at the wrong aggregate;
- Nº Failed Searches: Number of times the circle didn't find the aggregate (these include the Nº Deviation to Other Aggregates);
- Accuracy: Indirect variable calculated from the successful searches divided by the number of total searches (56 for this dataset);
- Otsu Time: Time it took the algorithm to run the otsu code (used as ac comparison validation metric)

Obtained Data - Brute Density Ascend Algorithm Version

Situation 1.1							
	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean	Error
Search Time (sec)	161.73	159.43	156.51	157.59	160.24	159.10	2.08
Nº Early Stops	34.00	34.00	34.00	34.00	34.00	34.00	0.00
Nº Deviation to Other Agreggates	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Nº Failed Search	56.00	56.00	56.00	56.00	56.00	56.00	0.00
Accuracy (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Otsu Time (sec)	3.80	3.42	3.34	3.09	3.00	3.33	0.31

Figure 1: Data from Horrible Start

Situation 1.2							
	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean	Error
Search Time (sec)	74.23	74.59	74.84	71.57	71.30	73.31	1.72
Nº Early Stops	7.00	7.00	7.00	7.00	7.00	7.00	0.00
Nº Deviation to Other Agreggates	4.00	4.00	4.00	4.00	4.00	4.00	0.00
Nº Failed Search	14.00	14.00	14.00	14.00	14.00	14.00	0.00
Accuracy (%)	75.00	75.00	75.00	75.00	75.00	75.00	0.00
Otsu Time (sec)	4.02	4.06	3.61	3.51	3.12	3.66	0.39

Figure 2: Data from Medium Start

Situation 1.3							
	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean	Error
Search Time (sec)	38.32	39.09	34.84	34.16	34.84	36.25	2.27
Nº Early Stops	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Nº Deviation to Other Agreggates	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Nº Failed Search	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Accuracy (%)	100.00	100.00	100.00	100.00	100.00	100.00	0.00
Otsu Time (sec)	4.81	3.33	3.44	3.06	3.12	3.55	0.72

Figure 3: Data from Perfect Start

Obtained Data - Optimized Density Ascend Algorithm Version

Situation 2.1							
	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean	Error
Search Time (sec)	16.52	18.64	18.62	14.26	11.70	15.95	2.98
Nº Early Stops	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Nº Deviation to Other Agreggates	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Nº Failed Search	56.00	56.00	56.00	56.00	56.00	56.00	0.00
Accuracy (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Otsu Time (sec)	3.56	4.38	4.10	3.66	3.05	3.75	0.51

Figure 4: Data from Horrible Start

Situation 2.2							
	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean	Error
Search Time (sec)	16.23	17.19	12.82	11.36	11.49	13.82	2.72
Nº Early Stops	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Nº Deviation to Other Agreggates	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Nº Failed Search	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Accuracy (%)	100.00	100.00	100.00	100.00	100.00	100.00	0.00
Otsu Time (sec)	3.76	3.83	3.43	4.15	3.87	3.81	0.26

Figure 5: Data from Medium Start

Situation 2.3							
	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean	Error
Search Time (sec)	10.31	9.58	9.14	9.72	10.03	9.76	0.45
Nº Early Stops	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Nº Deviation to Other Agreggates	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Nº Failed Search	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Accuracy (%)	100.00	100.00	100.00	100.00	100.00	100.00	0.00
Otsu Time (sec)	3.31	3.54	3.64	3.35	3.31	3.43	0.15

Figure 6: Data from Perfect Start

Obtained Data - Dynamic Density Ascend Algorithm Version

Situation 3.1							
	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean	Error
Search Time (sec)	18.53	14.72	13.76	16.10	14.06	15.43	1.95
Nº Early Stops	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Nº Deviation to Other Agreggates	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Nº Failed Search	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Accuracy (%)	100.00	100.00	100.00	100.00	100.00	100.00	0.00
Otsu Time (sec)	3.97	3.94	3.90	3.23	3.10	3.63	0.43

Figure 7: Data from Horrible Start

Situation 3.2							
	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean	Error
Search Time (sec)	13.96	15.42	12.04	12.22	11.07	12.94	1.73
Nº Early Stops	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Nº Deviation to Other Agreggates	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Nº Failed Search	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Accuracy (%)	100.00	100.00	100.00	100.00	100.00	100.00	0.00
Otsu Time (sec)	4.03	3.90	3.65	3.86	3.07	3.70	0.38

Figure 8: Data from Medium Start

Situation 3.3							
	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean	Error
Search Time (sec)	14.36	11.50	12.67	16.16	15.93	14.12	2.03
Nº Early Stops	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Nº Deviation to Other Agreggates	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Nº Failed Search	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Accuracy (%)	100.00	100.00	100.00	100.00	100.00	100.00	0.00
Otsu Time (sec)	3.38	3.96	3.79	3.68	3.68	3.70	0.21

Figure 9: Data from Perfect Start