UNIVERSIDAD SAN PABLO CEU

ESCUELA POLITECNICA SUPERIOR

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FINAL DE CARRERA

BLIND QUALITY ESTIMATION OF 3D MICROGRAPH RECONSTRUCTIONS

Alvaro Capell Osorio

Carlos Oscar S. Sorzano

Julio 2011

UNIVERSIDAD SAN PABLO-CEU

ESCUELA POLITÉCNICA SUPERIOR

División de Ingeniería Informática y de Telecomunicación

Calificación del Proyecto Fin de Carrera

Datos del alumno

NOMBRE:

Datos del Proyecto

TÍTULO DEL PROYECTO:

Tribunal calificador

PRESIDENTE:	FDO.:

SECRETARIO:	FDO.:

VOCAL:	FDO.:

Reunido este tribunal el ______/______/______, acuerda otorgar al Proyecto de Fin de Carrera presentado por Don______________________________la calificación de _________________.

Agradecimientos

Hace cinco años comencé mis estudios en la Universidad San Pablo CEU con más dudas que certezas. Al contrario que mucha gente nunca había sentido una vocación clara ni seguridad acerca del rumbo que quería imprimirle a mi vida. Cinco años después la satisfacción que me produce ser capaz de resolver problemas reales aplicando lo que he aprendido me ayuda a confirmar que no me equivoqué. Este cambio no habría sido posible sin la ayuda de los profesores de la universidad, a los que guardaré siempre gratitud por ello.

También me gustaría agradecer a Carlos Oscar su ayuda, sin la cual habría sido incapaz de llevar a buen puerto este proyecto, pero no ir un poco mas allá sería quedarme injustamente corto, porque además tengo que darle las gracias por proporcionarme unas ganas de aprender que no esperaba, por darme la oportunidad de desarrollarlas con este proyecto y por su tremenda paciencia y pedagogía cuando las dudas arreciaban.

Estoy donde estoy y soy lo que soy gracias a la gente que me rodea: Vicente y María, que me lo han dado todo, Diego, mi apoyo más cercano, Pablo, Marco, David, Over y María, que siempre han estado ahí, Edu y Dani, compañeros de batallas o Ana, mi estrella polar particular... Sin olvidarme de Donald, con el que no sólo he aprendido inglés sino que he crecido como persona. Sin estas raíces ni mis estudios ni este proyecto habrían sido posibles. A todos ellos, y a mucha mas gente que me dejo en el tintero... Gracias.

It is one of the commonest of mistakes to consider that the limit of our power of perception is also the limit of all there is to percieve.

C. W. Leadbeater

Resumen

La microscopía electrónica se ha convertido en una herramienta fundamental en la investigación biológica por su enorme capacidad de aumento en comparación con otras técnicas de microscopía tradicionales. Sin embargo, tales prestaciones tienen un precio, y es que las imágenes obtenidas mediante un microscopio de electrones contienen grandes cantidades de ruido. ¿Cual es el límite aceptable para ese ruido?. Y no menos importante ¿Cómo podemos estimar la calidad de una micrografía si no tenemos ninguna referencia con la que compararla?

El objetivo de este proyecto es precisamente ese: elaborar un programa que produzca una estimación sin referencias de la calidad de volúmenes de reconstrucción obtenidos mediante microscopía electrónica de manera que podamos saber de manera aproximada la resolución que poseen. Para ello se examinan una serie de técnicas propuestas por otros investigadores para la evaluación de la calidad de imágenes convencionales sin referencias y se detallan una serie de pruebas realizadas en Matlab para determinar si dichos métodos pueden aplicarse a los volúmenes 3D con los que trabajamos. Asimismo, se profundiza en el que consideramos más prometedor y se introducen una serie de modificaciones en el algoritmo para adaptarlo al trabajo en tres dimensiones, mejorar su precisión y robustecerlo frente al ruido.

En una última fase se implementa nuestro algoritmo en lenguaje C++, utilizando el entorno de desarrollo Eclipse y el paquete xmipp utilizado en la unidad de biocomputación del Centro Nacional de Biotecnología (CNB), y se realizan pruebas para comprobar que su funcionamiento es conforme a lo esperado.

Abstract

The overwhelming magnifying capacity that electron microscopy delivers has made this technique fundamental in biological investigation. However, such excellent performance has a drawback, as the images obtained using electron microscopes have high amounts of noise. What is the maximum amount of this noise we can accept? How can we grade the quality of a micrograph if we lack any reference for comparison?

This thesis works on that challenge: we want a program that can successfully produce a blind quality estimation index of a given 3D micrograph reconstruction volume, so that we can approximate the resolution of the volume. To do this we first examined various blind quality estimation methods for conventional images proposed by different researchers, and we also performed preliminary tests with Matlab to determine whether those techniques can be applied to our protein micrographs or not. After this we looked more deeply into the most promising algorithm and made some improvements to it, adapting it to 3D environments, upgrading its precision and strengthening it against noise.

Finally, we implemented this improved algorithm in C++ language, using the Eclipse development environment and the Xmipp package, used by the biocomputing unit at Centro Nacional de Biotecnología (CNB). A series of tests were then performed to verify that the final program fulfilled the goals we set.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Why electron microscopy?

Even though the Greeks and Romans already knew about the magnifying properties of pieces of glass and water orbs, and although primitive spectacles were used as early as the late 13^{th} century, we cannot properly talk about microscopy until the Renaissance. The earliest microscopes were mere tubes with lenses in one end, and, as the total magnification of the instrument did not exceed x10, these were used to entertain the public by viewing fleas and other small insects, and consequently named "flea glasses".

It is not certain who invented the first compound microscope, but most evidence points to Hans and Zacharias Janssen, two Dutch spectacle makers, as the first to experiment with several lenses in a tube. In 1609, Galileo heard of these experiments and formulated the principles of lenses, which helped him to produce a much better magni-

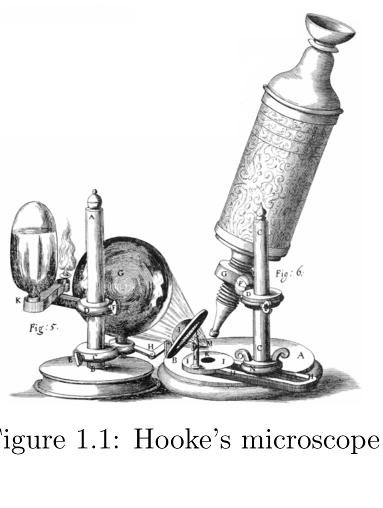


Figure 1.1: Hooke's microscope

fying instrument with one convex and one concave lens. In 1665, Robert Hooke published his *Micrographia*, a collection of biological micrographs. After this, the more refined lenses crafted by Anton van Leeuwenhoek would lead to a further breakthrough. But it was not until 1860, when Ernst Abbe discovered the Abbe sine condition, that trial and error was largely abandoned as a means of making improvements.

Present day instruments are much more finely crafted and can provide huge magnifications. However, even a microscope with perfect lenses and perfect illumination cannot discern objects smaller than half the wavelength of light: $0,275\mu$m. Such instruments are also extremely limited when observing transparent objects such as living cells, which have to be dyed before they can be studied, altering their original structure.

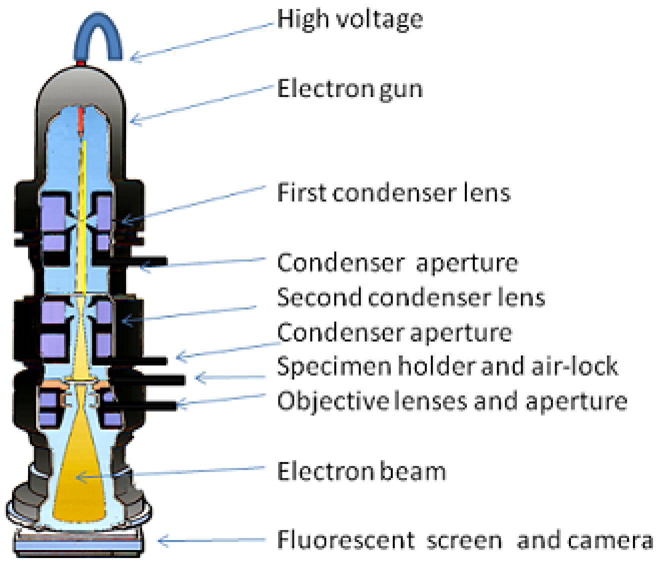


Figure 1.2: Transmission Electron Microscope

Thus, as the inherent limitations of optical microscopes are due to the wavelength of the photon, this was overcome by using electrons instead of photons. The wavelength of the electron is 100,000 times shorter than that of visible light, and so it can produce resolutions sharper than 50 pm. There are various kinds of electron microscopes; among the most important are Transmission Electron Microscopes (TEM, see 1.2) or Scanning Electron Microscopes (SEM).

In TEM microscopy a beam of electrons is transmitted through an extremely thin sample. First, the beam is generated from a source which can be made of tungsten or lantanum hexaboride. Then, it goes through a series of magnetic lenses that aim and focus it. Immediately afterwards, the electron stream interacts with a sample of the specimen to be analyzed. The magnetic lenses which follow amplify and correct the image, which will be recorded on the fluorescent screen and the camera at the bottom. The image of a mitochondria shown in 1.3 was obtained using a transmission electron microscope, as were all the images used in this thesis. The process occurs at near vacuum conditions, at pressures of $10^{-4} KPa$, to avoid air interfering with the electrons.

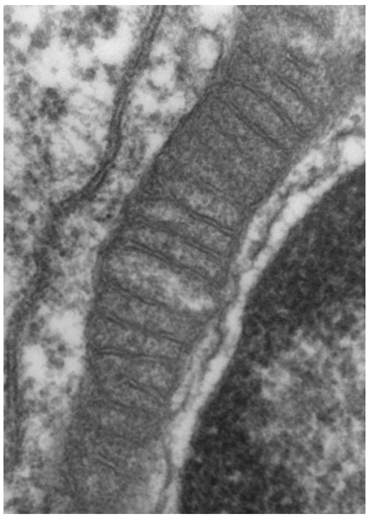


Figure 1.3: Mitochondria

One of the key elements of successful TEM analysis, as it is fairly complex, is the preparation of the sample (figure 1.4 shows the high complexity of an adequate TEM sample). In TEM, specimens must roughly measure a couple hundred nm wide for the electrons to go through them. Additionally, biological structures are extremely fragile, and inside the microscope they must withstand vacuum conditions and constant electron radiation. To protect them they can be fixed with a negative staining material, like uranyl acetate, or they can be embedded in vitreous ice at liquid nitrogen temperatures.

Just as the contrast of light microscope samples can be enhanced by using a light-absorbing dye, TEM samples' contrast can be enhanced by staining them with a metallic compound, such as gold. The metal will absorb or deflect electrons which would otherwise reach the fluorescent screen, greatly improving contrast. However, this also means that electrons are not allowed to penetrate inside the structures of the sample, limiting the information acquired to their surface. The metallic dye will also produce artifacts in the resulting image, ultimately limiting its maximum resolution to 20 Å.

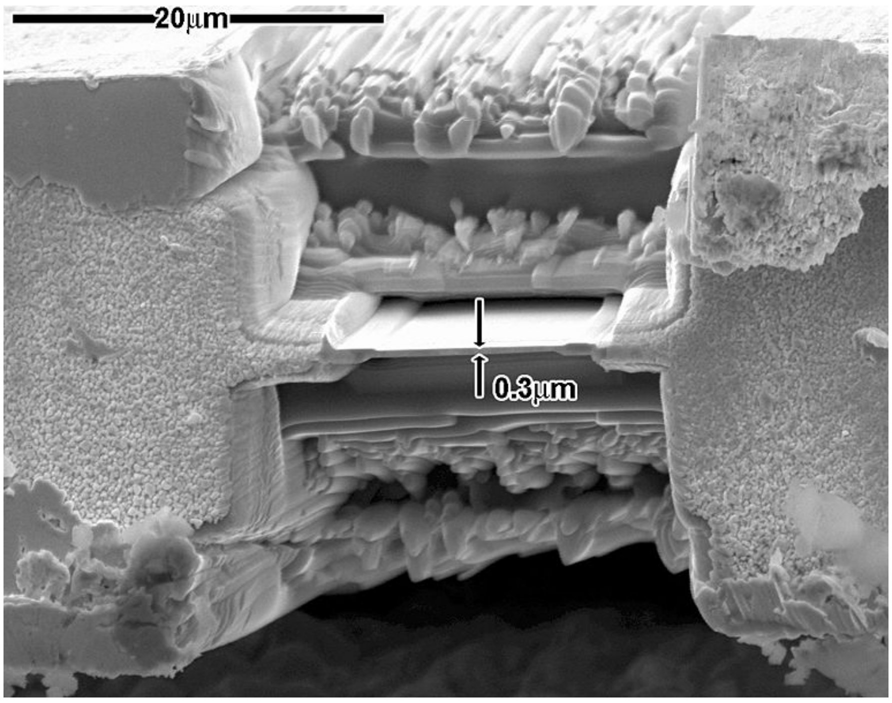


Figure 1.4: TEM sample

Cryo-electron microscopy is the alternative to the tincture. The sample is frozen at liquid nitrogen temperatures without it being stained or fixed, and thus, allowing observation in their native environment. This technique also allows deeper structural observation in samples, as electrons are no longer blocked by the dye in the surface of those structures. This is the most common way of preparing TEM samples nowadays.

1.2 Blind image quality estimation

Biological samples being extremely sensitive to radiation, electron exposure must be kept low to observe the sample without destroying it. This also means that the images obtained have a high amount of randomly distributed noise. There are several techniques that help to raise this signal to noise ratio, but even with these improvements noise is an extremely significant quality-limiting factor in the final resulting micrograph. Thus, evaluating the quality of the output image is crucial for determining wether it is valid for us to study it or should be discarded.

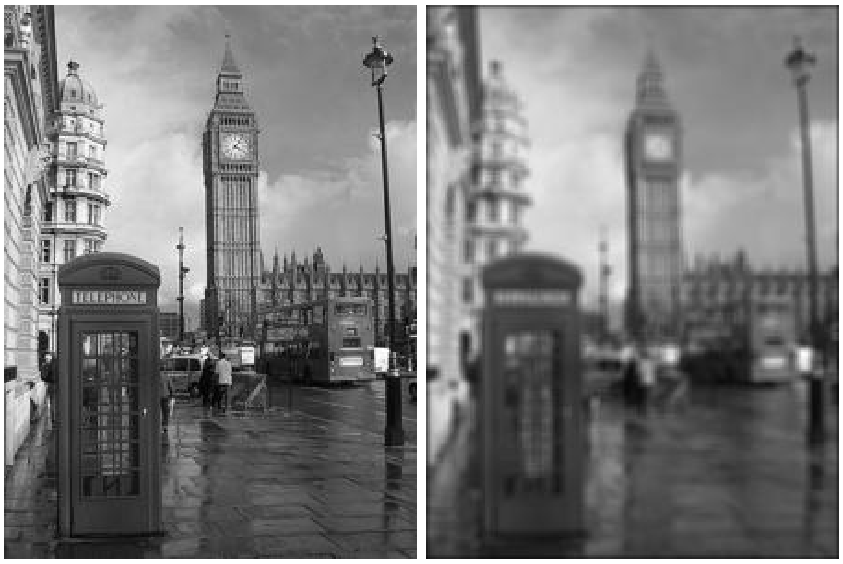


Figure 1.5: The same image with different qualities

Figure 1.5 shows two identical images of Big Ben. It is, however, obvious that the image on the left is the sharper. Comparing two identical images presents a problem which has quite a simple solution. Although a great number of algorithms can compare both pictures and select the best one, this is not the case with TEM micrographs: we do not have a flawless reference to compare with, nor a set of identical images from which to select the sharpest one. Only one image is available, and any estimation must be based on the mathematical properties of that image.

Blind image quality estimation is an extremely relevant subject nowadays, as well as a very active field for research: a multitude of theses, articles and studies have proposed new image quality assessment frameworks. In this thesis we have examined three different ways of estimating. One of the papers we have studied uses the blocking caused by JPEG compression to grade the quality of the image analyzed. Figure 1.6 shows the "blocking" phenomenon characteristic of this type of compression when the the ratio is too high.

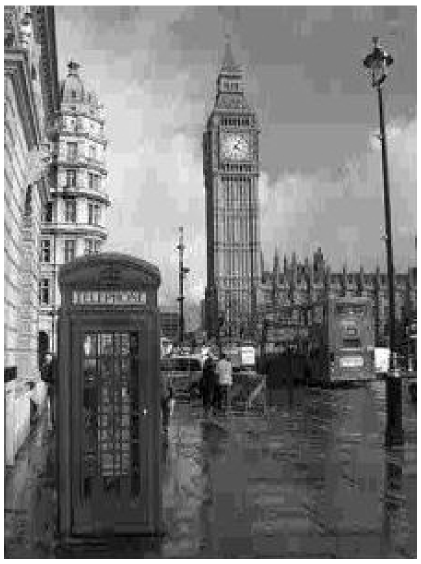


Figure 1.6: JPEG blocking

Another article proposes calculating the anisotropy histograms of consecutively blurred images to find possible patterns. The third, and ultimately successful, method introduces a new concept: "Just Noticeable Blur" (JNB), which consists of measuring the edge widths for every pixel of the image and returning the beta norm of all these values. Figure 1.7 shows the image edges, the actual edge width will be the amount of pixels between a pixel labeled as edge and the closest extrema. These methods are more profoundly analyzed and tested in Chapter 2.

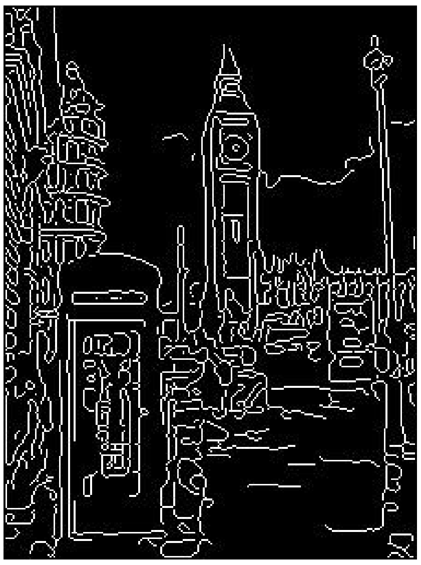


Figure 1.7: Edge detection

1.3 What were we aiming for?

Objectives

We want to develop software which can successfully determine whether 3D electron microscopy micrographs are sharp enough to be considered for their study with no other reference but the micrograph itself. The program will produce an index which, ideally, should have a linear relation to the actual resolution of the volume.

Procedure

To attain this objective we first analyzed papers with different blind image quality estimation algorithms. We chose three of these algorythms and performed some preliminary tests using the Matlab software kindly provided by the authors. The method developed by R. Ferzli and L. Karam stood out for the promising results it delivered, and so, using their idea, we developed a 3D version with B-spline interpolation, which is tailored for the specific 3D volume material we work with. To do this we have used C++ language with the Eclipse development environment and XMipp, the X-Windows-based Microscopy Image Processing Package by the Biocomputing Unit at CNB (Centro Nacional de Biotecnología).

Evaluation

Finally, we tested the new program using a set of volumes with decreasing resolution as inputs, and obtained a quality index for each one. These indexes were then analyzed to confirm wether the linearity we initially intended is met.

Chapter 2

Three ways to estimate image quality

2.1 No-Reference Perceptual Quality Assessment for JPEG Images

2.1.1 Introduction

Our first choice is the algorithm proposed by Z. Wang, H. R. Sheikh and A. C. Bovik in their article "No-reference perceptual quality assessment of jpeg compressed images" [1].

JPEG image compression algorithm provides a lossy compression method based on the Discrete Cosine Transform. The most noticeable effects of the loss of quality caused by this compression are blurring and blocking. This paper proposes an efficient way to extract features that can be used to reflect the relative magnitudes of these artifacts.

Denoting the test image signal as $x(m,n)\,\forall\, m \,\epsilon\, [1,M],\ n \,\epsilon\, [1,N]$, the algorithm proposed by Wang and his team has the following steps:

1. A differencing signal along each horizontal line is calculated:

$$d_h(m,n) = x(m,n+1) - x(m,n),\; n \,\epsilon\, [1, N-1]$$

2. Blockiness is measured in neighboring pixels in the boundaries of each 8x8 block:

$$B_h = \frac{1}{(\lfloor N/8 \rfloor - 1)} \sum_{i=1}^{M} \sum_{j=1}^{\lfloor N/8 \rfloor - 1} |d_h(i, 8j)|$$

3. The activity of the image is measured using two different factors:

   (a) First, the average absolute difference between in-block image samples:

   $$A_h = \frac{1}{7}\left[\frac{8}{M(N-1)} \sum_{i=1}^{M} \sum_{j=1}^{N-1} |d_h(i,j)| - B_h\right]$$

   (b) Second, zero-crossing rate is calculated:

   $$Z_h = \frac{1}{M(N-2)} \sum_{i=1}^{M} \sum_{j=1}^{N-2} z_h(m,n)$$

   where z_h is defined as:

   $$z_h = \begin{cases} 1 & \text{if horizontal ZC at } d_h(m,n); \\ 0 & \text{otherwise.} \end{cases}$$

4. The vertical features are calculated using analog methods, and the final overall features can be obtained:

$$B = \frac{B_h + B_v}{2}\;;\; A = \frac{A_h + A_v}{2}\;;\; Z = \frac{Z_h + Z_v}{2}$$

5. These features have to be combined in a way that constitutes a proper quality assessment model. The authors propose the following combination:

$$S = \alpha + \beta B^{\gamma_1} A^{\gamma_2} Z^{\gamma_3}$$

where parameters α , β , γ_1, γ_2 and γ_3 are calculated using a non-linear regression.

The score values range within 1 and 10, with 10 representing the best quality. It is possible that the output obtained for our image is not within this range, since the parameters are not properly trained for the type of images we will be evaluating. This is not a problem as we are only testing whether the quality index obtained is linear with the actual quality of the image. If the results are good enough we shall proceed to adapt this algorithm for the use with 3D volumes.

2.1.2 Testing

The first problem to be faced is that standard JPEG 8bit/pixel grey-scale images are assumed; that is, with a numerical value that varies within 0 and 255. Our images are not restricted by those limits and may even have negative values. Our data matrix must be rescaled before we begin with the testing:

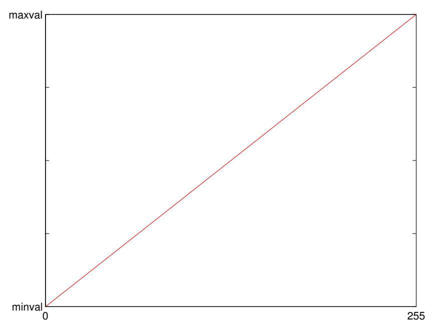
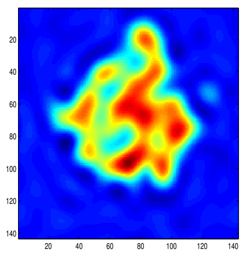


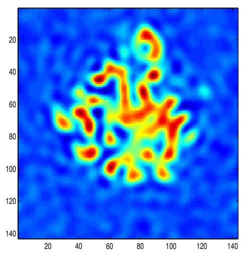
Figure 2.1: Re-scaling performed

Figure 1 shows the scaling we performed. A projection of the values between the minimum and the maximum to the interval 0-255 allows our rescaled matrix to fit to JPEG requirements: $y = 255 * \frac{x-minval}{maxval-minval}$

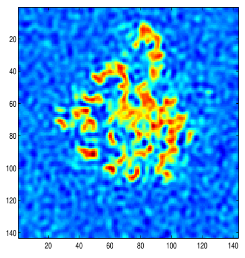
The algorithm is tested with each slice of the volume. The same slice is displayed below as an example, sorted in increasing resolution order:



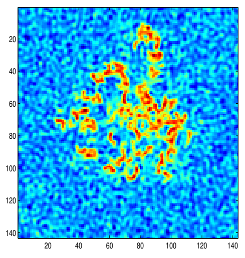
(a) 0.05 resolution



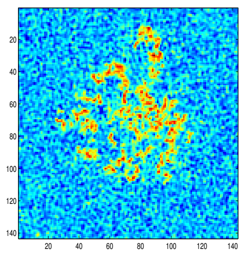
(b) 0.1 resolution



(c) 0.2 resolution



(d) 0.3 resolution



(e) 0.4 resolution

Figure 2.2: Samples from each batch of test pictures

Since each array is made slicing a 143x143x143 3D image volume, and every possible dimension is probed, the total amount of images of each sets will be 3x143=429. Quality index is measured for each individual picture and each set global quality is calculated as the average of them.

$$global\_index = \frac{\sum_{i=1}^{429} quality\_index(i)}{429}$$

For each volume file an average quality index was calculated:

Source	global_index
1FFKfull_noisy_0_05	74,8308
1FFKfull_noisy_0_1	39,4677
1FFKfull_noisy_0_2	21,8124
1FFKfull_noisy_0_3	29,4332
1FFKfull_noisy_0_4	33,5586

Table 2.1: Average quality index for different resolutions

2.1.3 Results

As we predicted, average results are not contained between 1 and 10, but range from 33 to 75. However, magnitude is not the most critical issue, it can be solved with a simple rescaling. Our aim is to find a linear relationship between the resolution and the image quality indexes given by the algorithm. Then we can adapt it to our specific images and grade their quality. As we can see in Table 2.1 this algorithm does not behave in the linear way we are searching. See Figure 2.3 for a graphical display of that information.

Results were quite promising for the first three volumes: the algorithm could sort them in increasing resolution order with no problem. We can see that in the linear behavior displayed in the first three points of the graph. However this tendency changed in the 4th and 5th batches, which were the highest resolution ones, but the algorithm failed to detect this improvement and delivered a higher value. This was probably caused because of the blurring produced by the lowpass filtering smoothened the noise in the empty spaces of the volume, and the algorithm interpreted this as a lower entropy sign, delivering a better quality estimation.

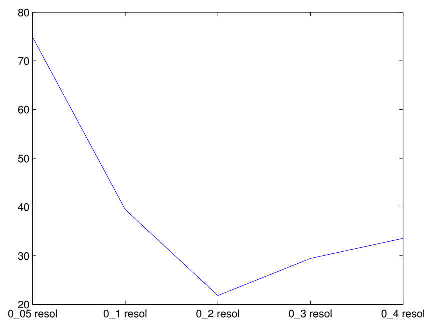


Figure 2.3: Quality Index/Resolution graph

In the real micrography analysis conditions we will be working with such low resolution samples are scarce, and this algorithm might never be moved out of its confidence region, but it is critical to ensure that if it happens our program is going to discard the volume. This means that we cannot make solid predictions based on the method proposed by Wang and his team, and thus, it has to be ruled out of the test list.

2.2 Quality Evaluation of Blurred and Noisy Images Through Local Entropy Histograms

2.2.1 Introduction

The next algorithm to be studied was proposed in the article "Quality Evaluation of Blurred and Noisy Images Through Local Entropy Histograms" by *G. Cristóbal* and *S. Gabarda* [2].

Shannon and Wiener proposed the definition of entropy as a measure of the information content per symbol coming from a stochastic source. This idea was later extended to yield a more general concept of entropy. Among these more general forms of entropy we find Rényi entropy which has the following form when applied to a discrete space-frequency distribution $P[n,k]$:

$$R_\alpha = \tfrac{1}{1-\alpha} \log \sum_n \sum_m P^\alpha[n,k]$$

where n is the spatial variable and k the frequency variable.

However, entropy is calculated over probability density functions. The space-frequency distribution is not a real probability density function, as it does not preserve either the positivity constraint $P[n,k] \geq 0$, nor the unity energy condition $\sum_k P[n,k] = 1$. Some normalization is therefore necessary.

1. To satisfy the first condition:

$$Q[n,k] = P[n,k] * P * [n,k] \ / \ Q[n,k] \geq 0 \ \forall n,k$$

2. To satisfy the second one:

$$\breve{P}[n,k] = \frac{Q[n,k]}{\sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} Q[n,k]} \; / \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \breve{P}[n,k] = 1$$

With this transformation, $\breve{P}[n,k]$ rows are formally correct probability density functions, and the notion of entropy is applicable. In this paper the authors propose to use the Renyi entropy ecuation shown in the figure above, with $\alpha = 3$ to build the mathematical model:

$$\breve{R}_3 = \frac{-1}{2}\log_2(\sum_n \sum_k \breve{P}^3[n,k])$$

By fixing the spatial variable a pixel-wise measure is obtained:

$$R[n] = \frac{-1}{2}\log_2(\sum_k \breve{P}^3[n,k])$$

Thus, the entropy result for each pixel depends on the space-frequency distribution associated to each pixel. For this particular problem, and inspired by quantum mechanics, the Pseudo Wigner Distribution (PWD) was chosen. The authors specifically used the Wigner distribution discrete aproximation proposed by Claasen and Mecklembräuker:

$$W[n,k] = 2\sum_{m=-\frac{N}{2}}^{\frac{N}{2}-1} z[n+m]z^*[n-m]e^{-2i(\frac{2\pi m}{N})k}$$

variable n represents time and variable k represents frequency. The shifting parameter m allows change of position and direction, but this movement is limited by N, which is the PWD window. In these tests N=8 is chosen.

As explained before, the value of R depends not only on the pixel which the calculation was made on, but also on the direction that was taken to calculate the PWD associated to each pixel. Six different directions are proposed:

$$\theta_s \, \epsilon \, [0^\circ, 30^\circ, 60^\circ, 90^\circ, 120^\circ, 150^\circ]$$

$R[n, \theta]$ is not a valid parameter to measure whole image characteristics since it depends on the individual pixel it is measured on. From this point on the mean average of the values obtained from all the pixels in the image will be used:

$$\overline{R}[\theta_s] = \frac{\sum_n R_3[n, \theta_s]}{M}$$

where M is the total amount of pixels of the image.The anisotropic quality index (AQI) is defined as the standard deviation of the resulting set of values:

$$\sigma = \sqrt[2]{\sum_{s=1}^{S} (\mu_k - \overline{R}[\theta_s])/S}$$

where μ_k is the mean of the values of $\overline{R}$ for each direction θ_s: $\mu_k = \frac{\sum_{s=1}^{S} \overline{R}[\theta_s]}{S}$

2.2.2 Applying the algorithm to our problem

For this test phase we will use the interface provided by the authors, which has the following work scheme: It will receive an image as input, generate a batch of 20 additional images by increasingly filtering or adding noise to the original image, show the profile of the anisotropy for each of the 21 images of the set, and provide a result of the AQI for the original image.

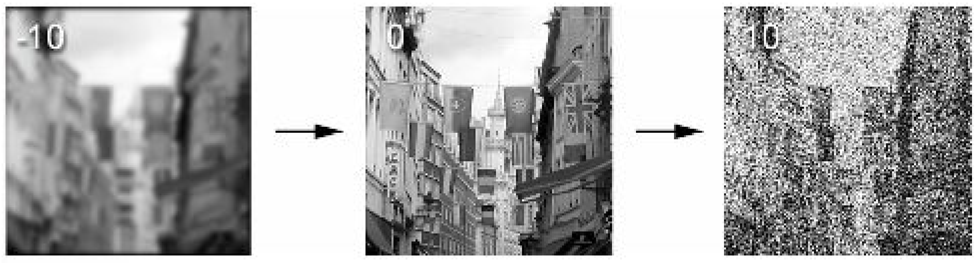


Figure 2.4: Algorithm test scheme

A typical profile produced would be similar to this:

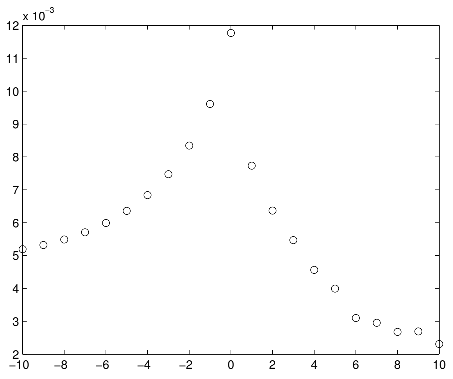
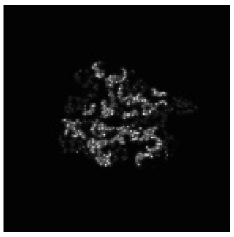


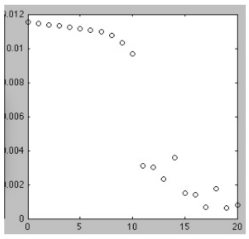
Figure 2.5: Example profile of AQIs

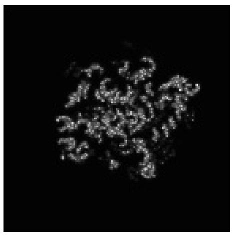
We applied this procedure to 6 notable frames between frame 50 and 100, in which the highest amount of information is concentrated. In the following pages each slice is displayed with its histogram. A table with the measured AQI is also shown for each volume.

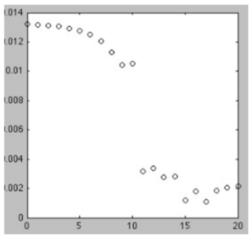
Frame	50	60	70	80	90	100
AQI	0.009684	0.010549	0.013124	0.012021	0.008663	0.0057073

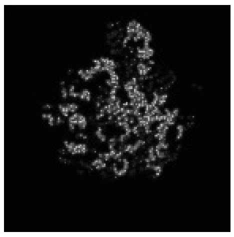
Table 2.2: AQI for 1FFKfull_0_5

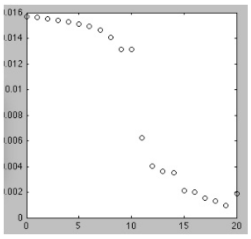

(a) Frame 50

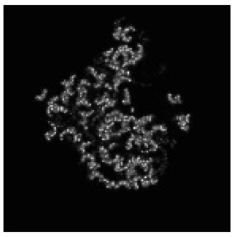

(b) Profile

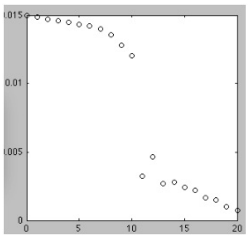

(c) Frame 60

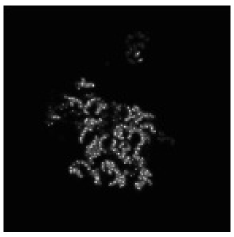

(d) Profile

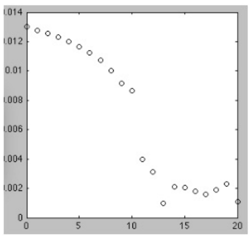

(e) Frame 70

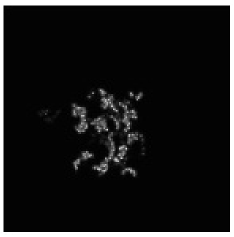

(f) Profile


(g) Frame 80


(h) Profile


(i) Frame 90


(j) Profile


(k) Frame 100

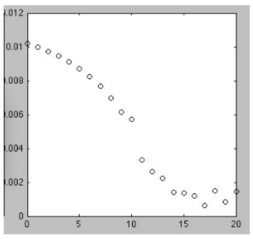
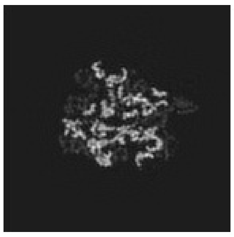
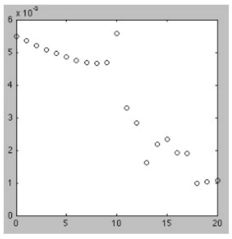

(l) Profile

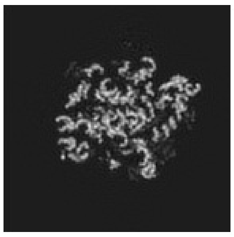
Figure 2.6: 1FFKfull_0_5

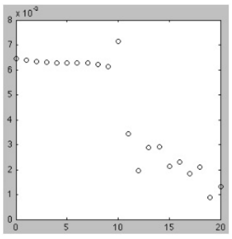
Frame	50	60	70	80	90	100
AQI	0.005565	0.007140	0.008344	0.008006	0.004791	0.003467

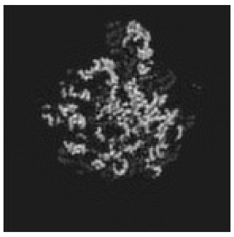
Table 2.3: AQI for 1FFKfull_0_4

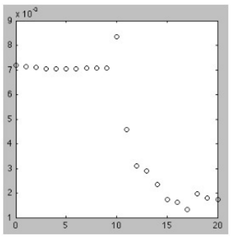

(a) Frame 50

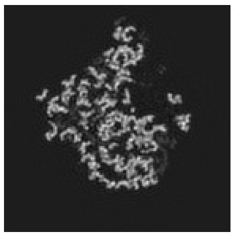

(b) Profile

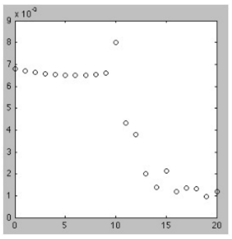

(c) Frame 60

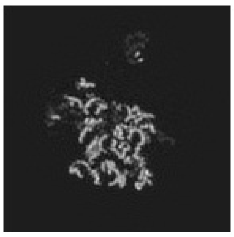

(d) Profile

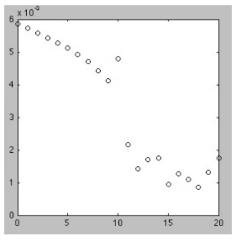

(e) Frame 70

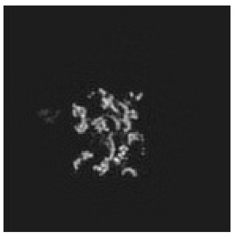

(f) Profile


(g) Frame 80


(h) Profile


(i) Frame 90


(j) Profile


(k) Frame 100

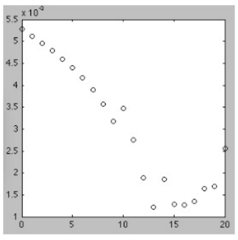
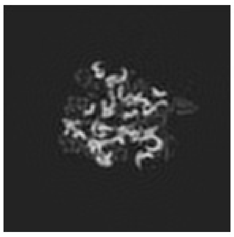
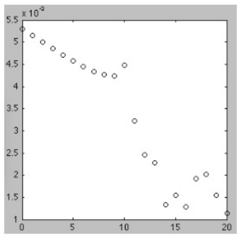

(l) Profile

Figure 2.7: 1FFKfull_0_4

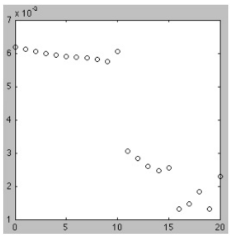
Frame	50	60	70	80	90	100
AQI	0.004479	0.006050	0.006941	0.006569	0.004023	0.002967

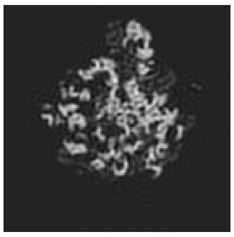
Table 2.4: AQI for 1FFKfull_0_3

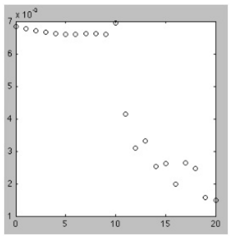

(a) Frame 50

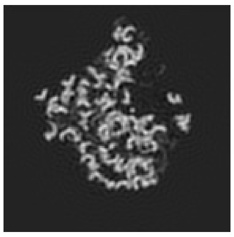

(b) Profile

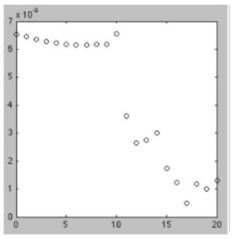
(c) Frame 60


(d) Profile

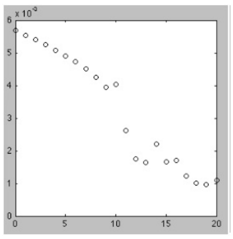

(e) Frame 70

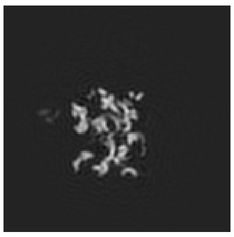

(f) Profile


(g) Frame 80


(h) Profile

(i) Frame 90


(j) Profile


(k) Frame 100

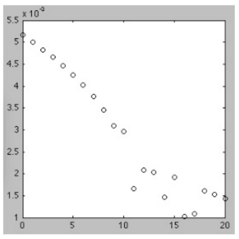
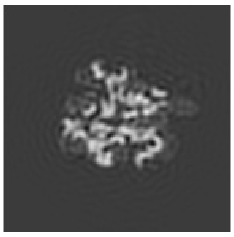
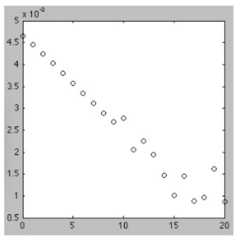

(l) Profile

Figure 2.8: 1FFKfull_0_3

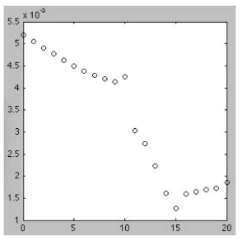
Frame	50	60	70	80	90	100
AQI	0.002773	0.004257	0.004678	0.004399	0.002649	0.002006

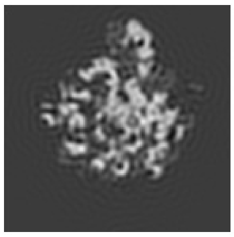
Table 2.5: AQI for 1FFKfull_0_2

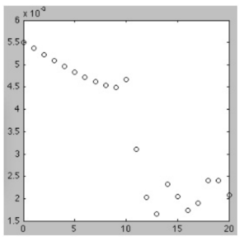

(a) Frame 50


(b) Profile

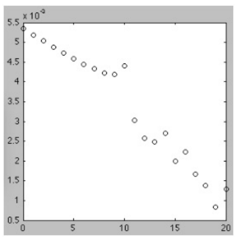
(c) Frame 60


(d) Profile

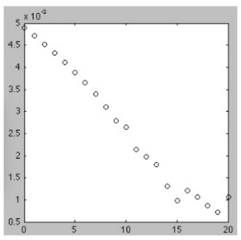

(e) Frame 70

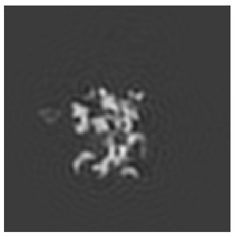

(f) Profile

(g) Frame 80


(h) Profile

(i) Frame 90


(j) Profile


(k) Frame 100

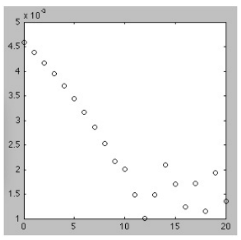
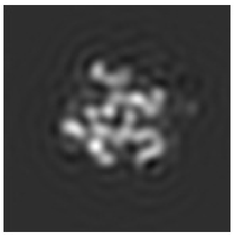
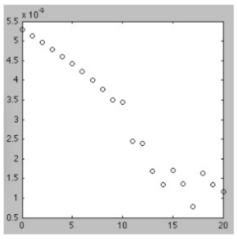

(l) Profile

Figure 2.9: 1FFKfull_0_2

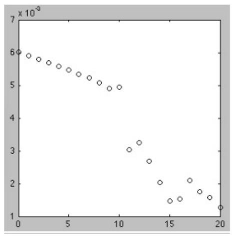
Frame	50	60	70	80	90	100
AQI	0.003445	0.004938	0.005973	0.005412	0.004250	0.002677

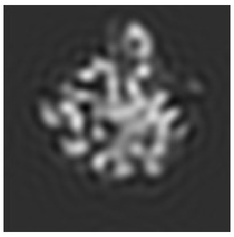
Table 2.6: AQI for 1FFKfull_0_1

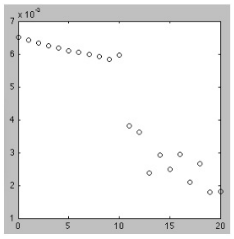

(a) Frame 50


(b) Profile

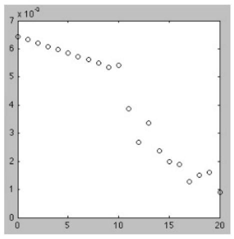
(c) Frame 60


(d) Profile

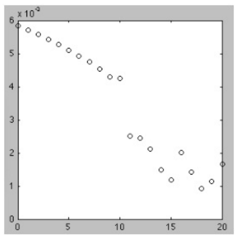

(e) Frame 70

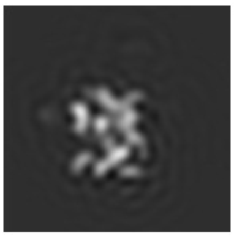

(f) Profile

(g) Frame 80


(h) Profile

(i) Frame 90


(j) Profile


(k) Frame 100

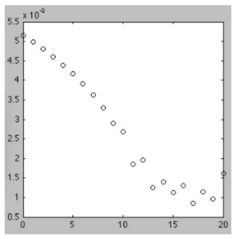
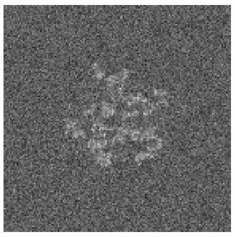
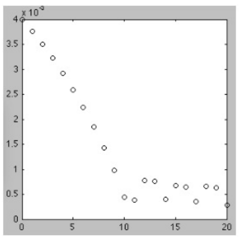

(l) Profile

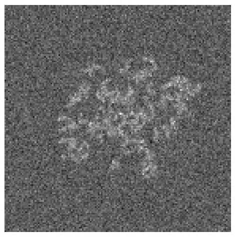
Figure 2.10: 1FFKfull_0_1

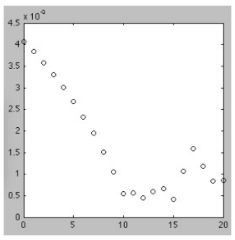
Frame	50	60	70	80	90	100
AQI	0.0004427	0.0005447	0.0008370	0.0005389	0.0004948	0.0004316

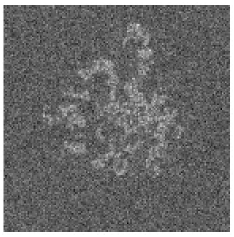
Table 2.7: AQI for 1FFKfull_noisy_0_5

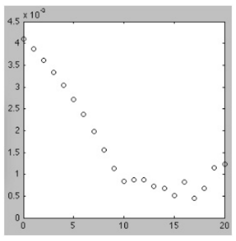

(a) Frame 50

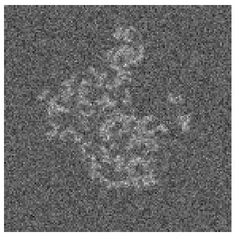

(b) Profile

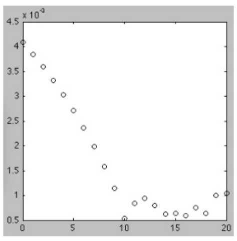

(c) Frame 60

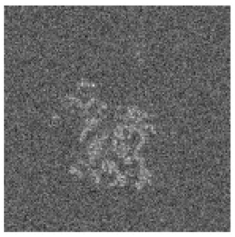

(d) Profile

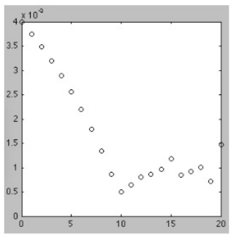

(e) Frame 70

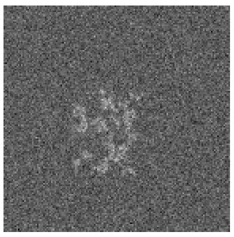

(f) Profile


(g) Frame 80


(h) Profile


(i) Frame 90


(j) Profile


(k) Frame 100

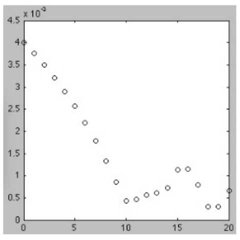
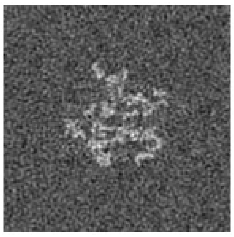
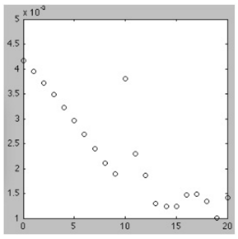

(l) Profile

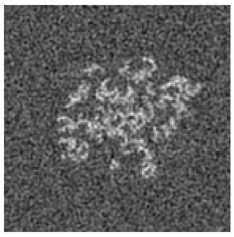
Figure 2.11: 1FFKfull_noisy_0_5

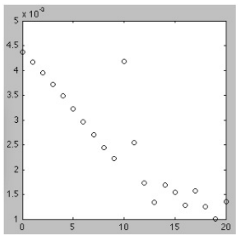
Frame	50	60	70	80	90	100
AQI	0.0038091	0.0041742	0.0045495	0.0042796	0.0036305	0.0034163

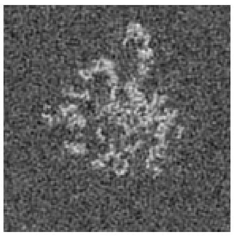
Table 2.8: AQI for 1FFKfull_noisy_0_4

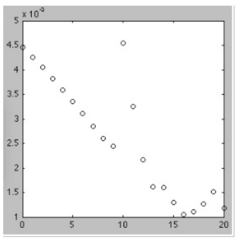

(a) Frame 50

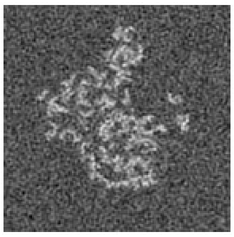

(b) Profile

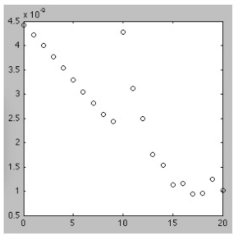

(c) Frame 60

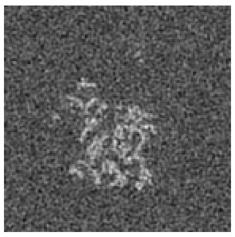

(d) Profile

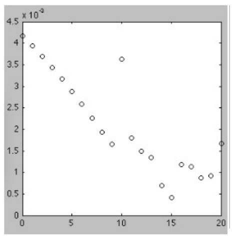

(e) Frame 70

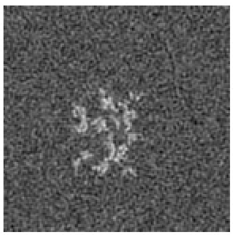

(f) Profile


(g) Frame 80


(h) Profile


(i) Frame 90


(j) Profile


(k) Frame 100

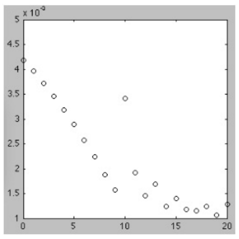
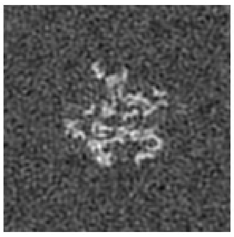
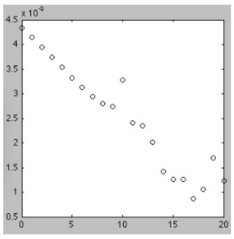

(l) Profile

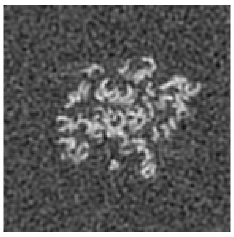
Figure 2.12: 1FFKfull_noisy_0_4

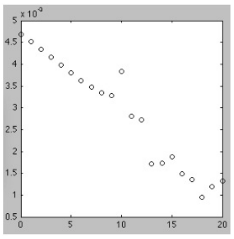
Frame	50	60	70	80	90	100
AQI	0.0032698	0.0038384	0.0044091	0.0043860	0.0030880	0.0028363

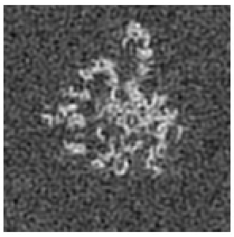
Table 2.9: AQI for 1FFKfull_noisy_0_3

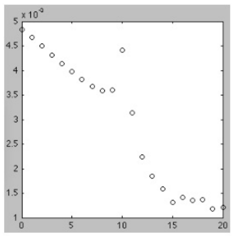

(a) Frame 50

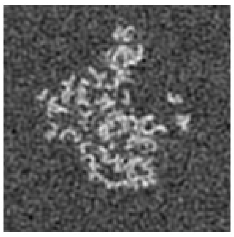

(b) Profile

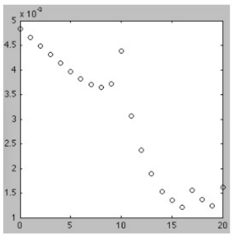

(c) Frame 60

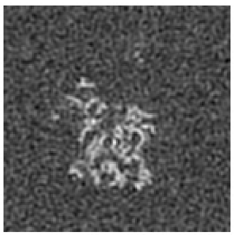

(d) Profile

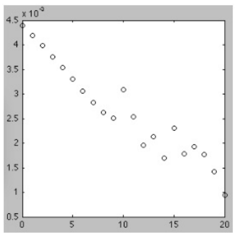

(e) Frame 70

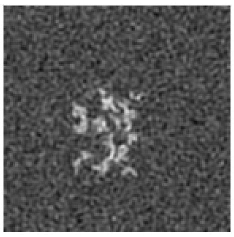

(f) Profile


(g) Frame 80


(h) Profile


(i) Frame 90


(j) Profile


(k) Frame 100

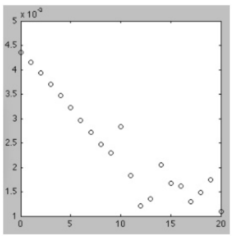
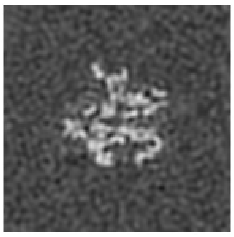
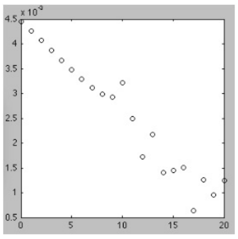

(l) Profile

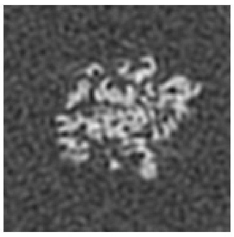
Figure 2.13: 1FFKfull_noisy_0_3

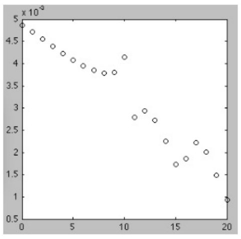
Frame	50	60	70	80	90	100
AQI	0.0032216	0.0041459	0.0044814	0.0045287	0.0031106	0.0025650

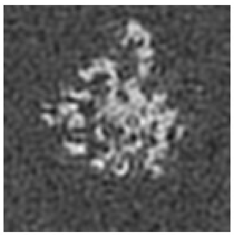
Table 2.10: AQI for 1FFKfull_noisy_0_2

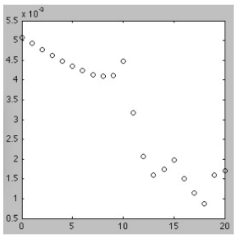

(a) Frame 50

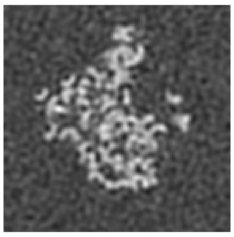

(b) Profile

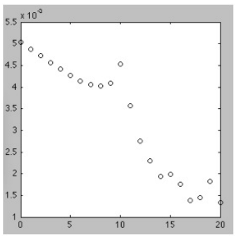

(c) Frame 60

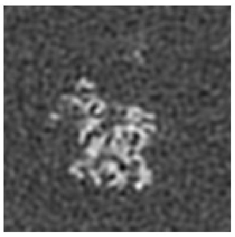

(d) Profile

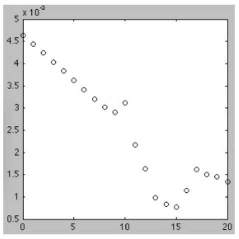

(e) Frame 70

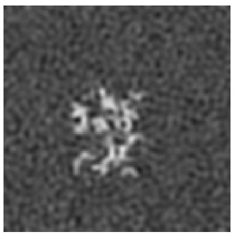

(f) Profile


(g) Frame 80


(h) Profile


(i) Frame 90


(j) Profile


(k) Frame 100

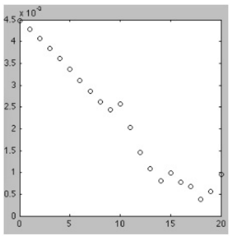
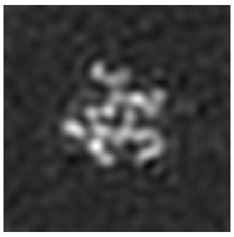
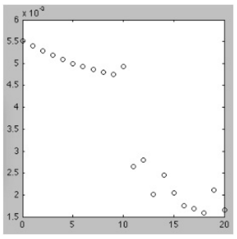
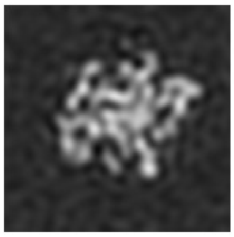
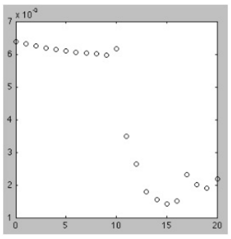

(l) Profile

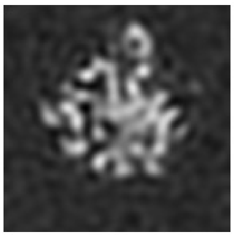
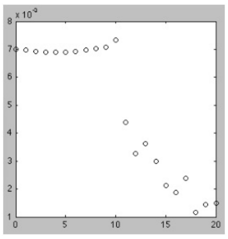
Figure 2.14: 1FFKfull_noisy_0_2

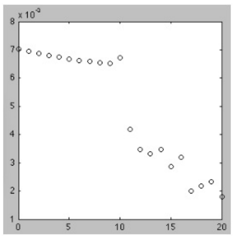
Frame	50	60	70	80	90	100
AQI	0.0049264	0.0061564	0.0073384	0.0067250	0.0054106	0.0038814

Table 2.11: AQI for 1FFKfull_noisy_0_1

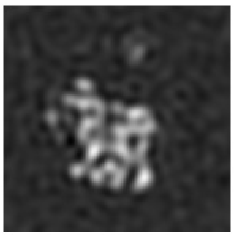
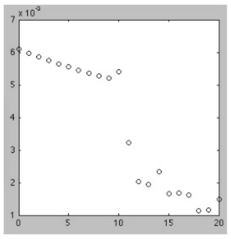
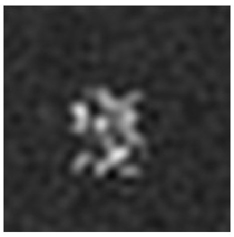
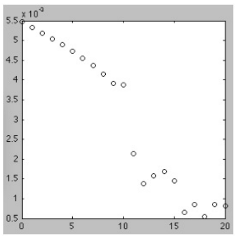





(a) Frame 50 (b) Profile (c) Frame 60 (d) Profile



(e) Frame 70 (f) Profile (g) Frame 80 (h) Profile

(i) Frame 90 (j) Profile (k) Frame 100 (l) Profile

Figure 2.15: 1FFKfull_noisy_0_1

2.2.3 Results

The following table has all the data gathered for the volumes to which gaussian noise was not added:

Resol	Frame 50	Frame 60	Frame 70	Frame 80	Frame 90	Frame 100
0.5	0.009684	0.010549	0.013124	0.012021	0.008663	0.0057073
0.4	0.005565	0.007140	0.008344	0.008006	0.004791	0.003467
0.3	0.004479	0.006050	0.006941	0.006569	0.004023	0.002967
0.2	0.002773	0.004257	0.004678	0.004399	0.002649	0.002006
0.1	0.003445	0.004938	0.005973	0.005412	0.004250	0.002677

Table 2.12: AQI for noiseless volumes

Although the algorithm successfully sorted the four highest resolution volumes by decreasing quality order, the one with the lowest resolution is not detected properly and a higher quality estimate is delivered:

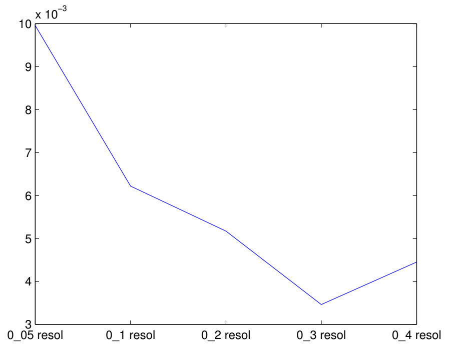


Figure 2.16: Noiseless performance

However, noiseless conditions are improbable: We must analyze the results for volumes which have gaussian noise for a sample of normal conditions. The following table displays all the data for the volumes that have gaussian noise added:

Resol	Frame 50	Frame 60	Frame 70	Frame 80	Frame 90	Frame 100
0.5	0.0004427	0.0005447	0.0008370	0.0005389	0.0004948	0.0004316
0.4	0.0038091	0.0041742	0.0045495	0.0042796	0.0036305	0.0034163
0.3	0.0032698	0.0038384	0.0044091	0.0043860	0.0030880	0.0028363
0.2	0.0032216	0.0041459	0.0044814	0.0045287	0.0031106	0.0025650
0.1	0.0049264	0.0061564	0.0073384	0.0067250	0.0054106	0.0038814

Table 2.13: AQI for noiseless volumes

The behavior of the algorithm when the input has gaussian noise added changes radically. We no longer see an inverse correspondence between the AQI and the resolution for the highest quality volumes:

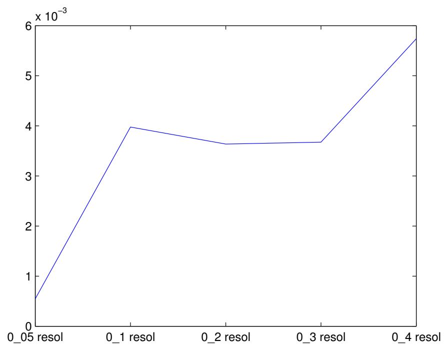


Figure 2.17: Noisy performance

Predictability is of critical importance for this work. We need our program to behave in a similar and predictable manner no matter what the input features are. It is not acceptable that noise interferes to the degree where detection is biased and a misleading result is provided.

2.2.4 Conclusion

After conducting these tests we noticed that the algorithm created by *G. Cristóbal* and *S. Gabarda* does not behave the same way regardless of the input. It was designed to analyze standard images, which are completely different from the micrographies we work with, and results are not sufficient for us to base our program on this method.

As with Wang's algorithm, we have to rule out this method as well.

2.3 A No-Reference Objective Image Sharpness Metric Based on the Notion of Just Noticeable Blur (JNB)

2.3.1 Introduction

After the two previous unsuccessful experiences we now try the method proposed in the work "A No-Reference Objective Image Sharpness Metric Based on the Notion of Just Noticeable Blur (JNB)" by R. Ferzli and L. J. Karam [3]. Instead of using JPEG compression phenomena or entropy, the authors propose the concept of just noticeable blur (JNB) and a probability summation model as a means of obtaining a no-reference sharpness metric.

Human visual sensitivity (HVS) cannot detect very subtle changes in the sharpness and contrast of an image, and so, up to a certain threshold, blurriness around an edge will be masked. This threshold is what the authors call JNB. However, since HVS varies from one individual to another subjective tests are required to determine the JNB depending on local contrast for the average individual. We are not going to explain these tests in detail, but the results are shown in the table displayed below:

$Contrast = \|foreground - background\|$	σ_{JNB}	ω_{JNB}
20	0.86	5
30	0.848	5
50	0.818	5
64	0.578	3
128	0.448	3
192	0.345	3
255	0.305	3

Table 2.14: Measured σ_{JNB} and ω_{JNB} for different contrasts

The σ_{JNB} magnitude is the standard deviation of the Gaussian filter that would produce precisely JNB blurring, and ω_{JNB} is the equivalent width in pixels for that σ.

The probability summation model that the authors propose considers a set of independent detectors placed on each edge pixel location noted as e_i. The probability of detecting blur distortion on a specific pixel is determined by the following function:

$$P(e_i) = 1 - exp(-|\frac{\omega(e_i)}{\omega_{JNB}(e_i)}|^{\beta})$$

Consequently, the probability of detecting blur distortion over a region of interest R is:

$$P_{blur}(R) = 1 - \prod_{e_i \epsilon R}(1 - P(e_i))$$

If both expressions are combined we get:

$$P_{blur}(R) = 1 - exp(-D_{(R)}^{\beta})$$

where $D_{(R)}$ is equivalent to:

$$D_{(R)} = (\sum_{e_i \epsilon R} |\frac{\omega(e_i)}{\omega_{JNB}(e_i)}|^{\beta})^{\frac{1}{\beta}}$$

The algorithm proposed uses image blocks of 64x64 pixels. First Sobel edge detector is run over the entire image, then the blocks are examined to determine whether they are smooth blocks or edge blocks. If a block is found to have a higher number of edge pixels than a threshold, it is considered to be an edge block. In the contrary case, the block is labeled as smooth and it will not be processed at all, as smooth blocks do not contribute to the blur perception. The probability of detecting blur distortion in the whole image will then be:

$$P_{blur}(I) = 1 - \prod_{R_b \epsilon I}(1 - P_{blur}(R_b))$$

where, as seen before, $R_{blur}(R_b)$ is:

$$P_{blur}(R_b) = 1 - exp(-D^{\beta}_{(R_b)})$$

Once again, combining both expressions yields:

$$P_{blur}(I) = 1 - exp(-D^{\beta})$$

where D stands for:

$$D = (\sum_{R_b} |D_{R_b}|^{\beta})^{\frac{1}{\beta}}$$

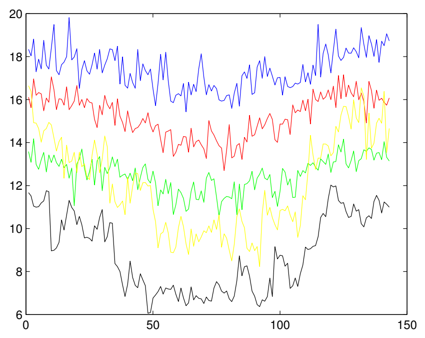
The resulting blur distortion measure D is then normalized by the amount of blocks labeled formerly as edge blocks, with the proposed no-reference objective sharpness metric being the following:

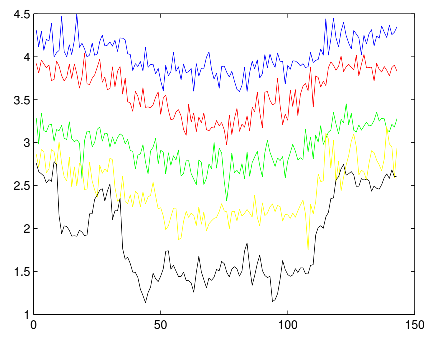
$$S = (\frac{L}{D})$$

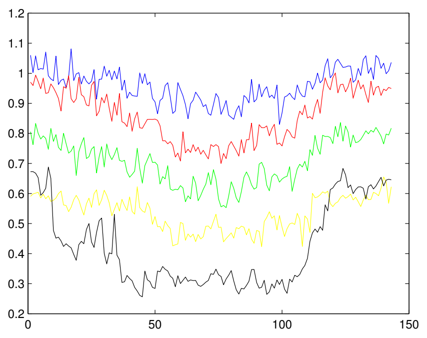
with L as the amount of processed blocks.

2.3.2 Tests

For this preeliminary test we used the Matlab software provided by Ferzli and Karam. However some slight modification had to be made: instead of picking a fixed size block of 64 we changed the code so that block size was an additional input parameter. We were then able to test block sizes of 143 (whole frame), 64, 32 and 16 for each resolution. The test procedure was therefore similar to the one used with Wang's algorithm, with the volume sliced into its 143 frames and the JNBM_compute function called to calculate quality index estimation for each frame. This was done for each block size and each resolution.


(a) 16x16 block size


(b) 32x32 block size


(c) 64x64 block size

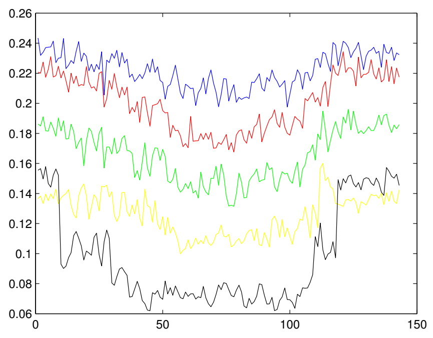

(d) 143x143 block size

Figure 2.18: Quality estimation results for each block size and resolution

The results of these calculations can be found in Figure 2.18 (in decreasing resolution order: blue, red, yellow, green and black):

These preeliminary findings show that results are better for bigger block sizes than for smaller ones. We then chose to set the block size to 143x143 (the whole frame) as execution time was substantially smaller. However, we noticed that for 0.1 resolution unexpected behavior occurs on the most outer frames:

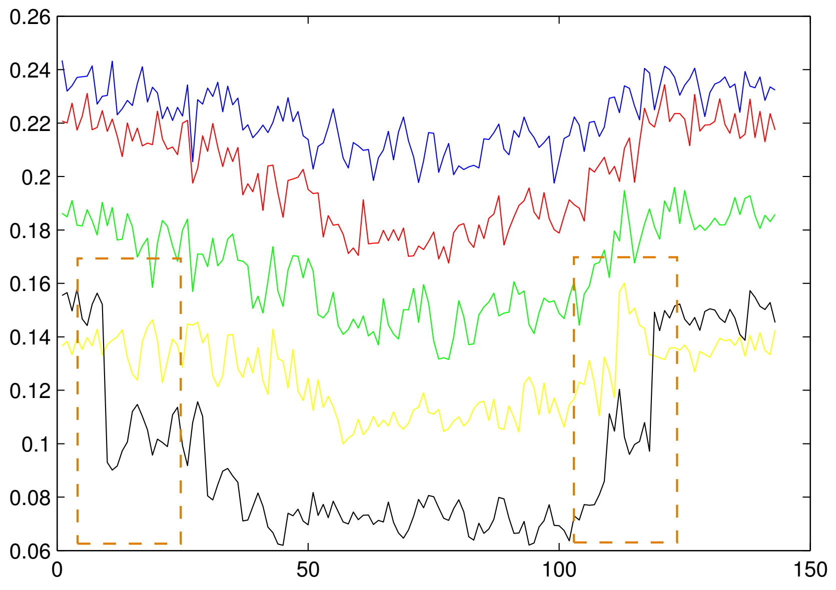


Figure 2.19: Lowest resolution behavior

The quality index provided for 0.1 resolution rises rapidly in the frames which border the protein, surpassing even the values obtained for the same frames of the 0.2 resolution volume. This behavior is due to the way the probability of detecting blur in the image is obtained:

$$P(e_i) = 1 - exp(-|\frac{\omega(e_i)}{\omega_{JNB}(e_i)}|^{\beta})$$

ω_{JNB} values can be found in 2.14. Between contrast values of 50 and 64, ω_{JNB}

varies from 3 to 5. For all the resolutions except 0.1 ω_{JNB} remains constant as 3, but in those specific frames of the 0.1 resolution volume, contrast value is lower than 50, so ω_{JNB} changes to 5 and quality estimation value increases suddenly.

To deal with this minor flaw we have made one additional slight modification in the code.

```
widthjnb = [5*ones(1,60) 3*ones(1,30) 3*ones(1,180)];
```

Instead of using the ω_{JNB} values provided we substituted them for a constant array, so that no undesired changes would occur regardless of the contrast value:

```
widthjnb = [3*ones(1,270)];
```

The same tests were performed again, and we obtained the following values:

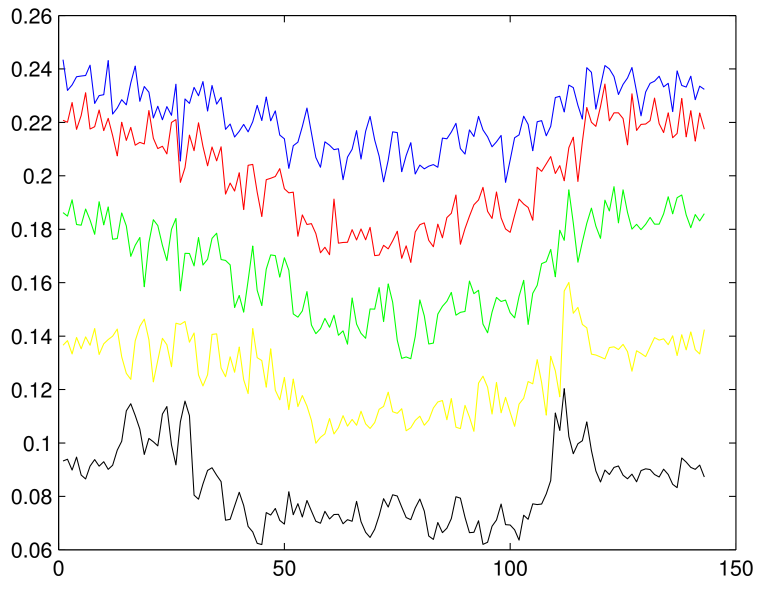


Figure 2.20: Smooth behavior

2.3.3 Results

As seen in 2.21 the correspondence between resolution and the quality index is strongly linear. If we calculate the average of all the results obtained for each resolution we can see this linearity very clearly:

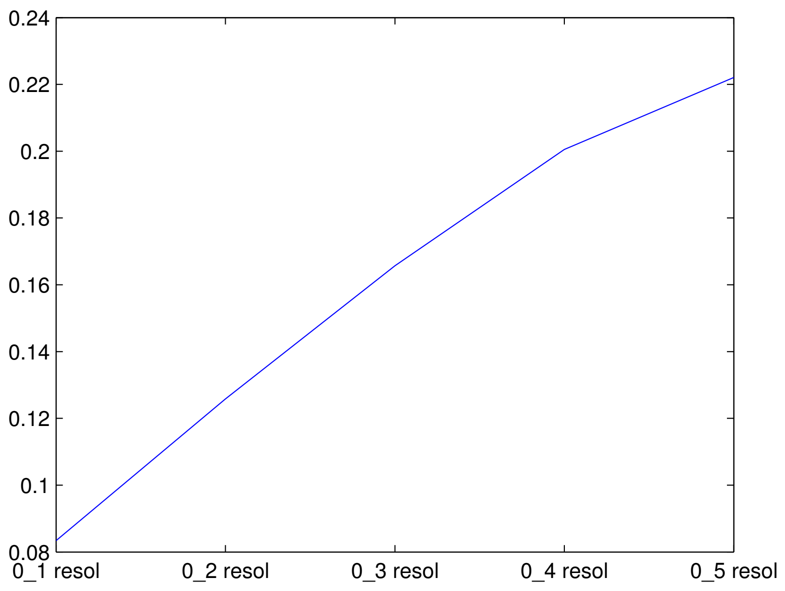


Figure 2.21: Linear behavior

Based on these results we can estimate image quality based on the results of the JNB prediction which will have a very strong correspondence with the resolution of the volume analysed. We believe that these results are conclusive enough to stop searching and base our program on this method.

Chapter 3

Our version of the algorithm

3.1 Introduction

In the previous section we examined different methods of obtaining blind image quality estimations, we explained their mathematical formulation and we performed several quick tests to find out whether they were useful for solving our problem or not. These preliminary results showed that the image quality estimation method described in the work by R. Ferzli and L. Karam was extremely promising for determining the quality of our 3D micrography reconstruction volumes.

However, as it was merely a preliminary test phase, only a rough analysis was made. In this section we will examine this in depth, describing the requirements which had to be met as well as other necessary modifications, and lastly the final version of the algorithm will be explained.

3.1.1 Summary of the algorithm by Ferzli and Karam

As we previously saw, R. Ferzli and L. Karam studied the minimum amount of blurriness around an edge that the average individual would discern, calling this concept Just Noticeable Blur (JNB). First, this was quantified as being the standard deviation of the Gaussian filter which would produce such blurring, and afterwards this was converted to its equivalent in pixels: ω_{JNB}. However, the actual widths of the edges in the picture must be measured in order to compare them with JNB width. As we can see in Figure 3.1, which is directly taken from their work, R. Ferzli and L. Karam propose searching for the local extrema immediately before and after the pixel being analyzed.

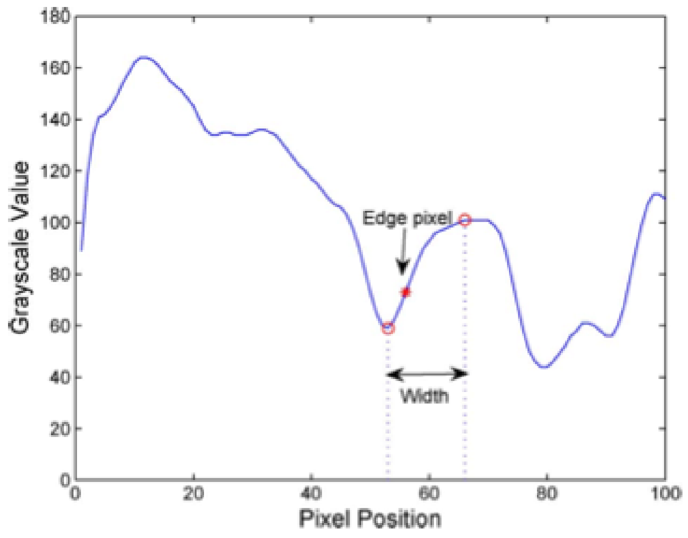


Figure 3.1: Edge width

Having introduced these two concepts, the probability of detecting blur in a specific point e_i is, as seen in chapter 2:

$$P(e_i) = 1 - exp(-|\frac{\omega(e_i)}{\omega_{JNB}(e_i)}|^{\beta})$$

Which, extended to a certain region R yields the following expression:

$$P_{blur}(R) = 1 - \prod_{e_i \epsilon R} (1 - P(e_i))$$

If both expressions are combined we get:

$$P_{blur}(R) = 1 - exp(-D_{(R)}^{\beta})$$

where $D_{(R)}$ is equivalent to:

$$D_{(R)} = (\sum_{e_i \epsilon R} |\frac{\omega(e_i)}{\omega_{JNB}(e_i)}|^{\beta})^{\frac{1}{\beta}}$$

To find the probability of detecting blur over a whole image, we have to extend this expression to the entire number of regions comprised by the image:

$$P_{blur}(I) = 1 - \prod_{R_b \epsilon I} (1 - P_{blur}(R_b))$$

which can be simplified to:

$$P_{blur}(I) = 1 - exp(-D^{\beta})$$

where D stands for:

$$D = (\sum_{R_b} |D_{R_b}|^{\beta})^{\frac{1}{\beta}}$$

The resulting parameter D, normalized by the total number of regions, is the proposed quality estimation metric. Figure 3.2 explains how this algorithm works visually.

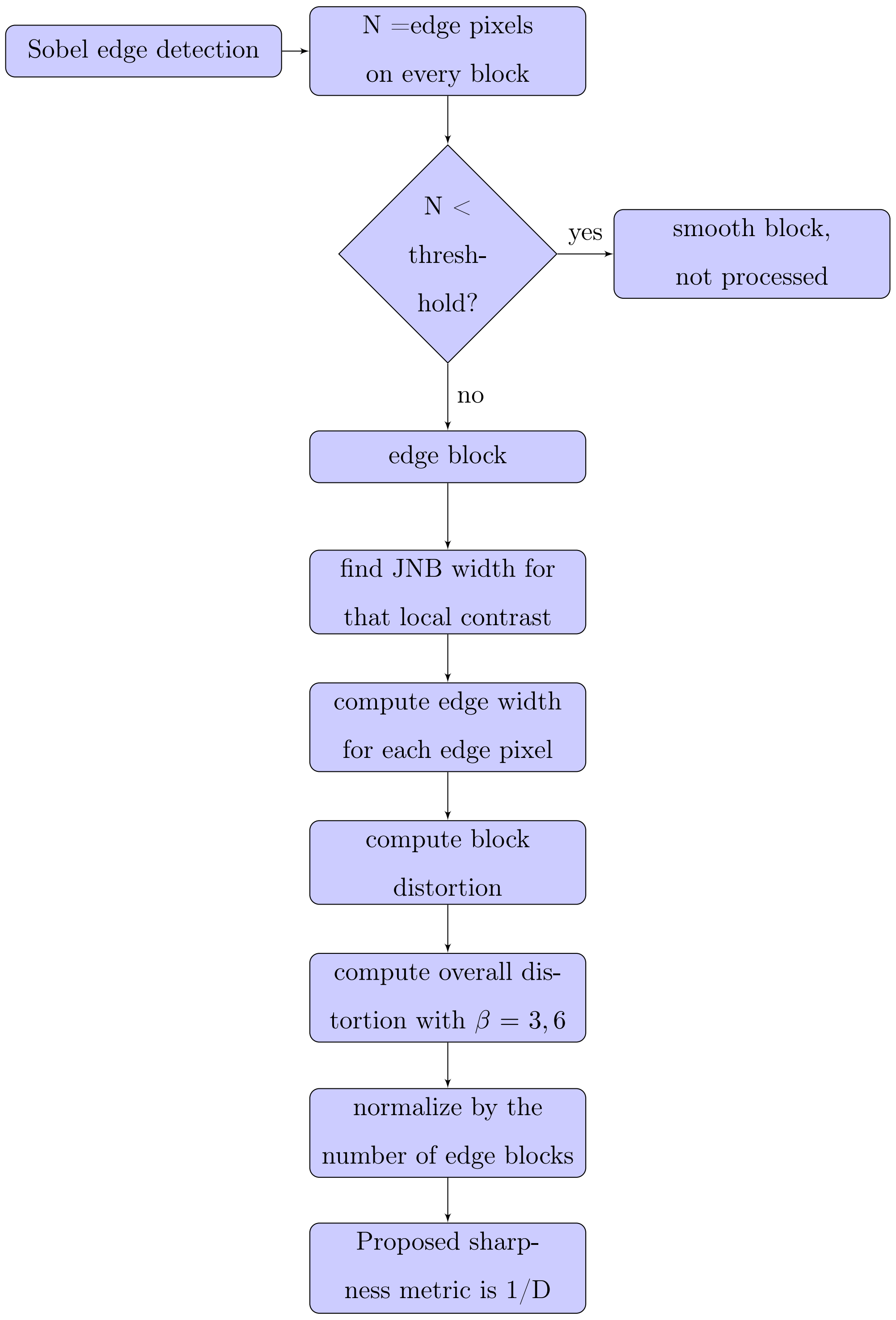

Sobel edge detection
N =edge pixels on every block
N < thresh-hold?
yes
smooth block, not processed
no
edge block
find JNB width for that local contrast
compute edge width for each edge pixel
compute block distortion
compute overall dis-tortion with β = 3, 6
normalize by the number of edge blocks
Proposed sharp-ness metric is 1/D

Figure 3.2: Flowchart

3.1.2 Simplifications made

As explained in the preceding chapter, we had to make the JNB width parameter constant to ensure that the linear relationship between resolution and quality index was preserved. By examining the references in Ferzli and Karam's paper we were led to the work "A no-reference perceptual blur metric" by P. Marziliano, F. Dufaux, S. Winkler and T. Ebrahimi [4], which proposed a similar way to measure blur, but in a much simpler fashion.

Marziliano et al eliminate the JNB usage and say that blur measure should be the sum of all edgewidths present in the image, divided by the number of edge pixels. A flowchart of the algorithm is displayed below:

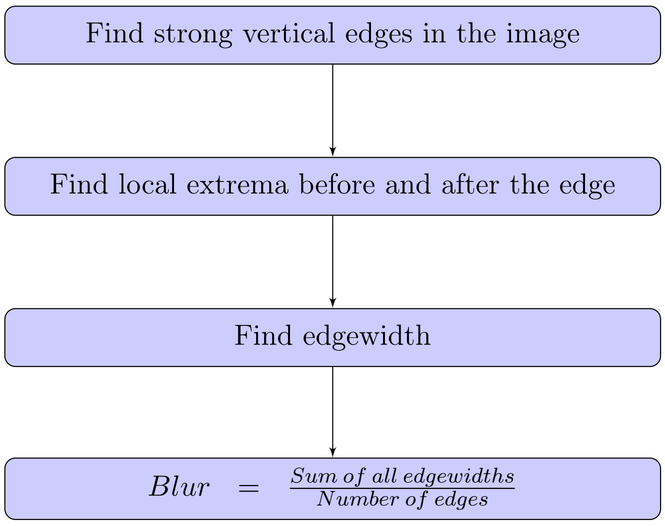

Find strong vertical edges in the image
Find local extrema before and after the edge
Find edgewidth
$Blur = \frac{Sum\ of\ all\ edgewidths}{Number\ of\ edges}$

Figure 3.3: Yet another flowchart

Note that this metric does not divide the image into blocks, but processes it as a whole. This agrees with the experimental results we obtained in Chapter 2 as well, and so in our implementation we will process entire images rather than blocks.

3.2 Requirements

3.2.1 Full 3D point of view

It was stated previously that the methods we studied are aimed at 2D images. For the test phase we simply divided the micrograph volume into slices and treated each one as an individual image, but this is not good enough for the final algorithm. We need full 3D processing to guarantee optimal results and this was achieved through a twofold effort.

First, as the only-vertical approach seen in the methods we examined is sufficient for 2D images but not for 3D volumes, we have taken an additional step and now search for edge widths in $\vec{i}$, $\vec{j}$, $\vec{k}$ and additional directions, which are represented in Table 3.1. We believe that processing volumes in these directions allows us to consider their 3D features and provides a sound algorithm performance. The calculated widths associated to edge pixels will be stored in an auxiliary volume. As these widths will be calculated successively for each different direction, this volume will only store the minimum width associated to a pixel. If the calculated width for a particular direction is higher, it will be discarded.

Vectors
(1, 0, 0)
(0, 1, 0)
(0, 0, 1)
$(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)$
$(\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}})$
$(0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$
$(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}})$
$(\frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}})$
$(\frac{1}{\sqrt{3}}, \frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}})$
$(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{-1}{\sqrt{3}})$

Table 3.1: Directions

Second, as prior to this edge width search the protein edges must be enhanced by some edge detection algorithm, such as Sobel. Since this method is mainly focused towards 2D images, we need a version which is compatible with 3D processing. To overcome this we relied heavily on Fernando Fuentes's masters thesis: "Single particle electron microscopy volume restoration" [5]. In this work he developed a Sobel version in which he substituted the partial derivatives with B-Spline derivatives in x, y and, as an improvement for 3D processing, z as well. We will explain what B-Splines are in the next section.

3.2.2 B-Spline interpolation

Conventional images are typically several hundreds if not thousands of pixels wide. The methods we tested take steps of 1 pixel in each iteration, which compared to the overall size of the image are relatively small and consequently produce only minor errors. However, our 3D reconstruction volumes hardly measure 150 pixels wide, which makes a 1 pixel leap unacceptably large compared to total size. This high step-size ratio might lead to significant errors during volume processing; it is thus necessary to find a way to take smaller steps. This way is interpolation, and more specifically, spline interpolation

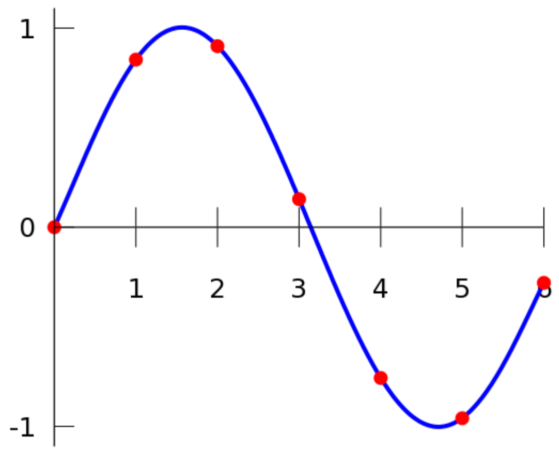


Figure 3.4: Polynomial interpolation

Interpolation consists of estimating new data points within the range of a discrete set of known data points. There are several kinds of interpolation, among which we find linear, polynomial or spline, to name only a few. As the name implies, linear interpolation considers the interpolant to be a straight line that joins the two data points together. This is a quick and easy way of interpolating, but not very precise and has the massive disadvantage of producing non-differentiable outputs.

A more sophisticated approach is polynomial interpolation: here, straight lines are substituted for polynomials, which can vary in degree. For n data points there is exactly one polynomial of degree $n-1$ which goes through all the points. However, this process is quite expensive computationally, and the output might be flawed by oscillatory artifacts due to Runge's phenomenon.

To avoid these disadvantages (or at least dramatically reduce them) we can use spline interpolation. Splines are piecewise polynomials where the pieces are smoothly connected together. The fit might be exact or approximate depending on whether we choose interpolating splines or least-squares splines, and they vary as well in degree, from the most simple continuous representations to the most complex ones. Among the splines the most used are B-splines due to the high computational efficiency provided by their short support, as it is known that B-splines have the minimal support for a given approximation order. The following mathematical calculations are done in a single-dimensional space. To extend them to 2D tensor products must be considered:

$$\varphi_{2D}(x,y) = \varphi_{1D}(x)\cdot \varphi_{1D}(y)$$

Polynomials are not the only functions capable of interpolating. Whittaker proved that, if the x_i data points are regularly distributed, the series

$$C(x) = \sum_{i=-\infty}^{\infty} y_i sinc(\frac{x-x_i}{T})$$

also interpolates the input measurements (sinc is defined as $sinc(x) = \frac{sen(\pi x)}{\pi x}$). $C(x)$ is known as cardinal function and, as Shannon stated, it is unique for any function whose maximum frequency is smaller than $\frac{1}{2T}$ Hz. This is not only true for bandlimited functions, as Shannon's sampling theorem can be extended to a larger space of functions, such as the Hilbert space of L_2 functions or even all functions that are square-integrable in the Lebesgue sense. Consequently, the generalized sampling theorem is defined as:

$$C(x) = \sum_{i=-\infty}^{\infty} c_i \cdot \varphi_i(x)$$

where c_i are coefficients computed from the input data and $\varphi_i(x)$ is a shifted version of a basis function $\varphi(x)$.

For bandlimited functions the basis function used is $\varphi(x) = sinc(\frac{x}{T})$, which is the most widely known thanks to the sampling theorem. However, *sinc* decays very slowly and the convolutions needed for the inner product are impractical, so a more computationally attractive φ is in order. The shortest valid function is the box function:

$$\varphi(x) = \beta_0(\frac{x}{T}) = \begin{cases} 1 & |x| < \frac{T}{2} \\ 0 & |x| > \frac{T}{2} \end{cases}$$

which is the cardinal B-spline of degree 0.

To obtain the cardinal B-spline of degree n we only have to convolve $\beta_0(x)$ with itself n times. Consequently, β_1 will be a triangular function, and β_2 a parabolic function Their equations can be found below, assuming T=1:

$$\beta_1(x) = \begin{cases} 1 - |x| & |x| \leq 1 \\ 0 & |x| > 1 \end{cases}$$

$$\beta_2(x) = \begin{cases} \frac{3}{4} - |x|^2 & |x| \leq \frac{1}{2} \\ \frac{1}{2}(|x| - \frac{3}{2})^2 & \frac{1}{2} < |x| \leq \frac{3}{2} \\ 0 & |x| > \frac{3}{2} \end{cases}$$

$$\beta_3(x) = \begin{cases} \frac{2}{3} + \frac{1}{2}|x|^2(|x|-2) & |x| \le 1 \\ \frac{1}{6}(2-|x|)^3 & 1 < |x| \le 2 \\ 0 & |x| > 2 \end{cases}$$

In general, the cardinal B-spline of degree n can be expressed as:

$$\beta_n(x) = \frac{1}{n!}\sum_{k=0}^{n+1}(-1)^k \binom{n+1}{k}\left(x - \left(k - \frac{n+1}{2}\right)\right)_+^n$$

where $(x)_+^n$ is the one sided n-th power of x, $(x)_+^n = \begin{cases} n^n & x \ge 0 \\ 0 & x < 0 \end{cases}$

The representation of signals using cardinal B-splines is closely related with interpolation. For instance, the use of β_0 is equivalent to nearest-neighbor interpolation, while the use of β_1 is very similar to linear interpolation. β_3 provides the best compromise between complexity and approximation error. It can be proven that the following expression is an interpolating spline:

$$\varphi_{int}(x) = \sum_{i=-\infty}^{\infty} q_{int}[i]\varphi_i(x)$$

where $q_{int}[i]$ is the l_2 sequence defined as the inverse Z transform of $Q_{int}(z) = \frac{1}{\sum_{k=-\infty}^{\infty}\varphi(kT)z^{-k}}$ and $\varphi(x) = \beta_n(\frac{x}{T})$

3.2.3 Noise immunity

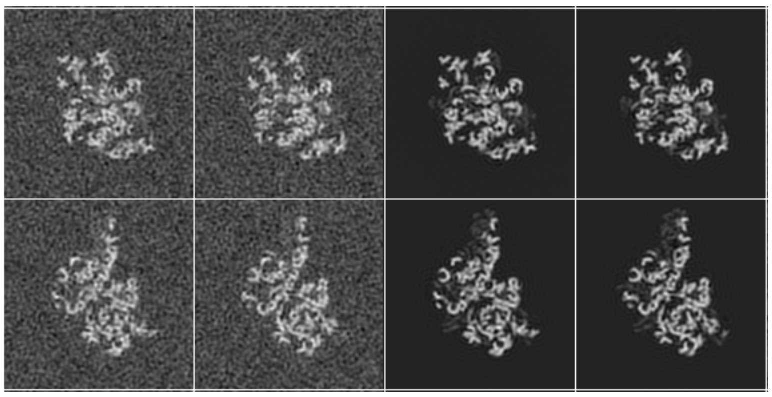


Figure 3.5: Volume frames where noise can be perceived

Noise adds an amount of uncertainty to any source of information, and this is as undesirable as it is inevitable. Our software has to withstand the high amount of noise inherent to images obtained through electron microscopy, without significantly varying the output from that which an ideal noiseless source would produce. Figure 3.5 shows true noisy slices compared with ideal noiseless ones.

It is obvious that such an amount of noise would pose a problem, and it did when running Sobel edge detection, as the function failed to discern which parts were valid data and which were discardable background (see Figure 3.6).

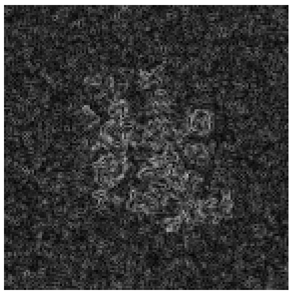


Figure 3.6: Noise effect

Again, Fernando Fuentes examined this issue in his master's thesis, and he programmed a background detection function that, given a reconstruction volume, generates a mask which allows background areas to be identified. Using this mask we can successfully eliminate background noise, leaving only the noise that affects the molecule areas, which is simpler to compensate for.

Thus, for the given molecule the following mask was calculated:

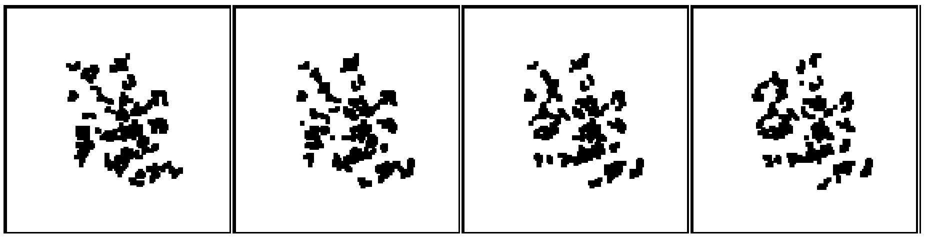


Figure 3.7: Calculated mask

This was done in a 3 step operation:

- First, confidence regions are calculated using the following distribution:

$$\frac{x - \mu_X}{\frac{S_X}{\sqrt{N_X}}} \equiv t_{N_x - 1}$$

- Second, pixels are processed taking into account those regions to detect whether they are noise or not. The planes which conform the borders of the volume are considered to be noise and the rest of the pixels are analyzed inwards from those planes to the central pixels.

- Third, to avoid isolated pixels, a morphology closing operation is performed. This merges all areas together into a single unified mask.

Figure 3.7 shows the original mask, but, as the selection threshold was slightly restrictive, a dilation was applied, and thus we got:

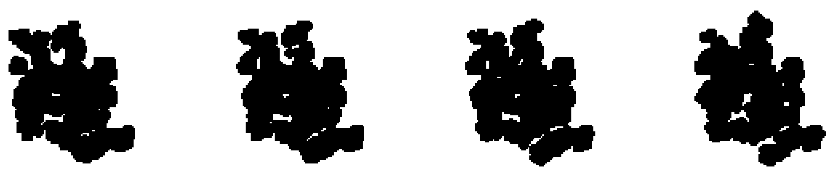


Figure 3.8: Dilated mask

If we apply that mask to the noisy edge volume we get:

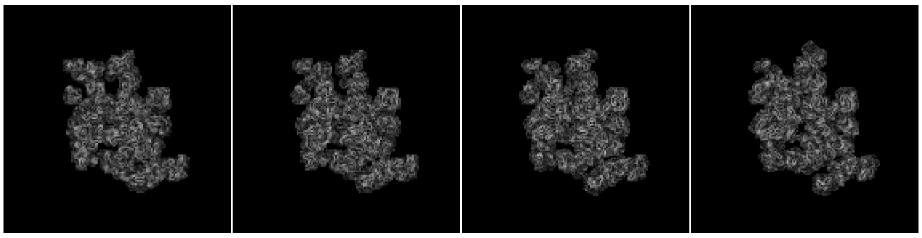


Figure 3.9: Masked edge volume

The difference is substantial, and this is even clearer if we compare the noisy preprocessed slices directly with the ones which have been masked:

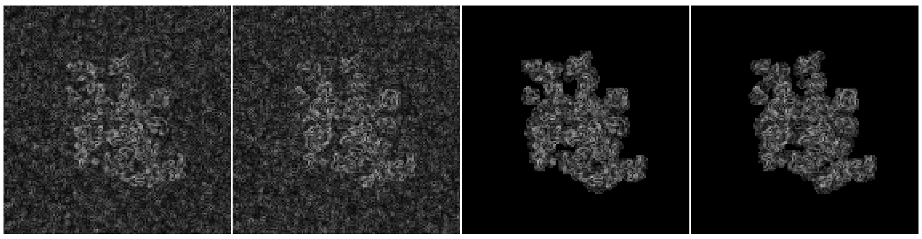


Figure 3.10: Noisy frames vs masked frames

We believe that this masking process makes our algorithm extremely robust against the noise produced by electron microscopy. In the next section we explain how our algorithm actually works.

3.3 Our implementation

To summarize, our program has to be capable of processing the full 3D features of our volumes, using B-spline interpolation to enable resolutions sharper than 1 pixel, as well as being relatively noise-proof. Even though it is built as a single function, we can discern two distinct parts in our program. The first is aimed at finding which pixels are relevant edge pixels, while the second's objective is to find the edge width associated to each of those pixels.

3.3.1 First part

The following flowchart corresponds to the first part, and will give the reader a rough idea of its mechanics before this is explained in depth later:

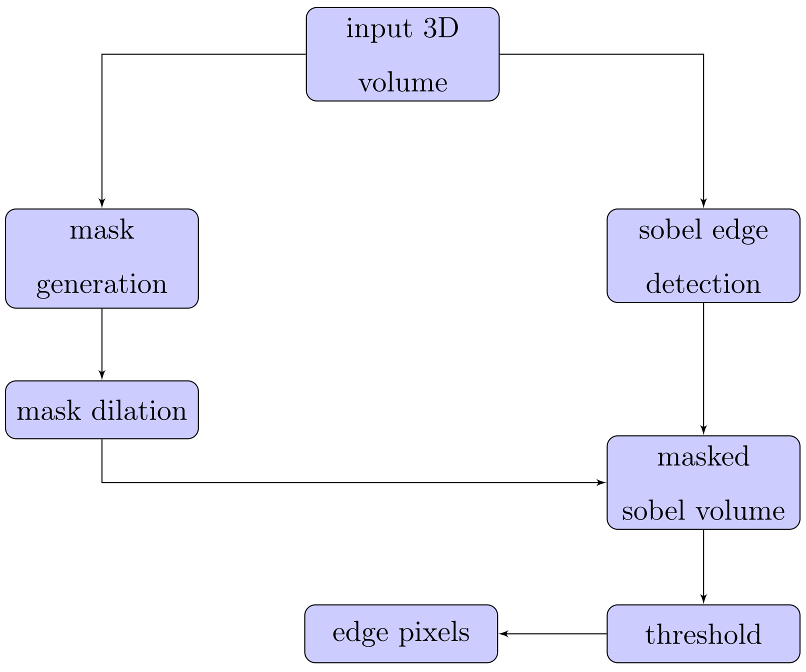


Figure 3.11: Edge pixel search

After the input volume is introduced, two operations run in parallel: on one side we run Fuentes's background detect function to calculate the mask needed to eliminate background noise, and on the other side we run Sobel edge detection function. This enhances the edges in the original volume, but is highly affected by noise. This can be seen in in Figure 3.12.

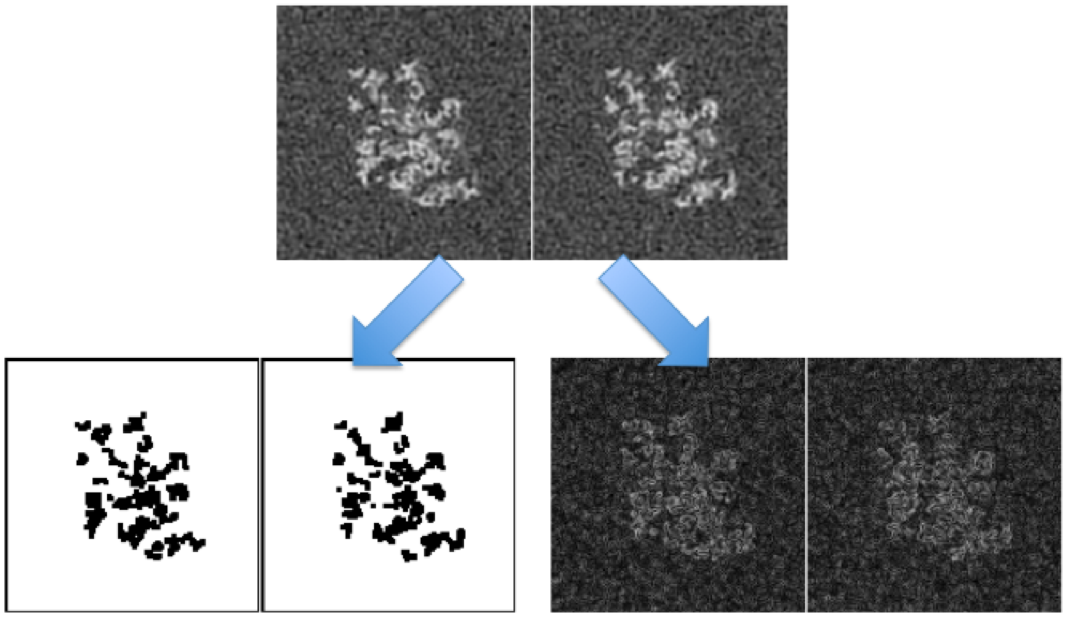


Figure 3.12: Mask calculation and edge enhancing

As explained in the last section, the mask here is too restrictive and, as well as removing the background, it might also eliminate valuable areas close to the edges of the molecule, which are less intense but interesting nonetheless. To prevent this from happening we apply a dilation to this mask, and allow more pixels to get through the masking operation.

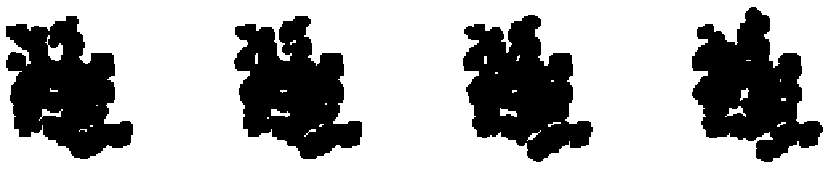


Figure 3.13: Dilated mask

The next step is to mask the enhanced-edge volume with our newly dilated mask. This allows all the relevant information-related pixels to remain unaltered while the background noisy ones are eliminated.

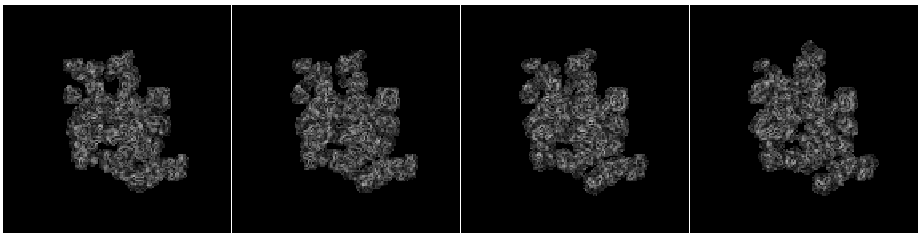


Figure 3.14: Masked edge volume

It remains to be decided which of these pixels are truly the edge pixels of the molecule. To do this a threshold is established. If the numerical value of the pixel exceeds this threshold, it is considered an edge pixel; otherwise, it is ignored.

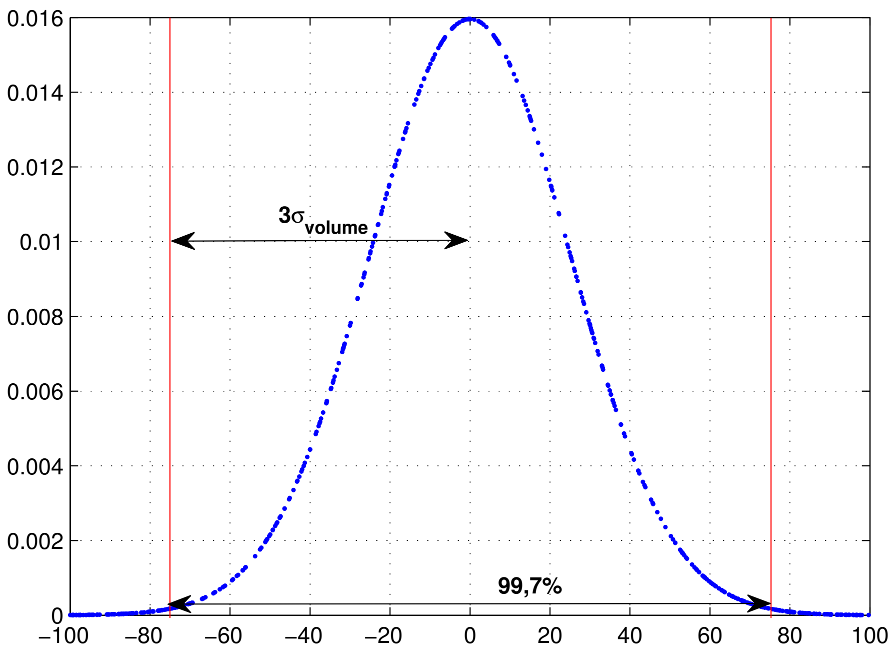


Figure 3.15: Threshold explanation

We determined experimentally that a threshold equal to three times the standard deviation of the values in the volume was appropriate, as 99,7% of the possible numerical grayscale values are included in the interval. Figure 3.15 shows a Gaussian distribution of random values in the interval $(-100, 100)$ with a mean of $\mu = 0$ and a standard deviation $\sigma = 25$.

After this, if the value of the pixel in the masked edge volume was superior to the threshold it was tagged as edge. Otherwise it was set to 0. This was done for all the pixels in the volume and the results can be seen in Figure 3.16

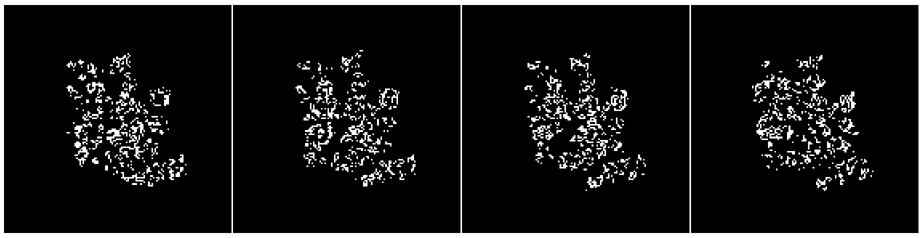


Figure 3.16: Edge pixels volume

White pixels are edge, the rest are non-edge. The next part of the program will use this volume to determine which pixels have to be processed and which do not.

3.3.2 Second part

This part's task is the calculation of the edge width associated to every pixel tagged as edge in the first part. This is done with the function edge_width. A rough idea of the way this function works is displayed in the following chart:

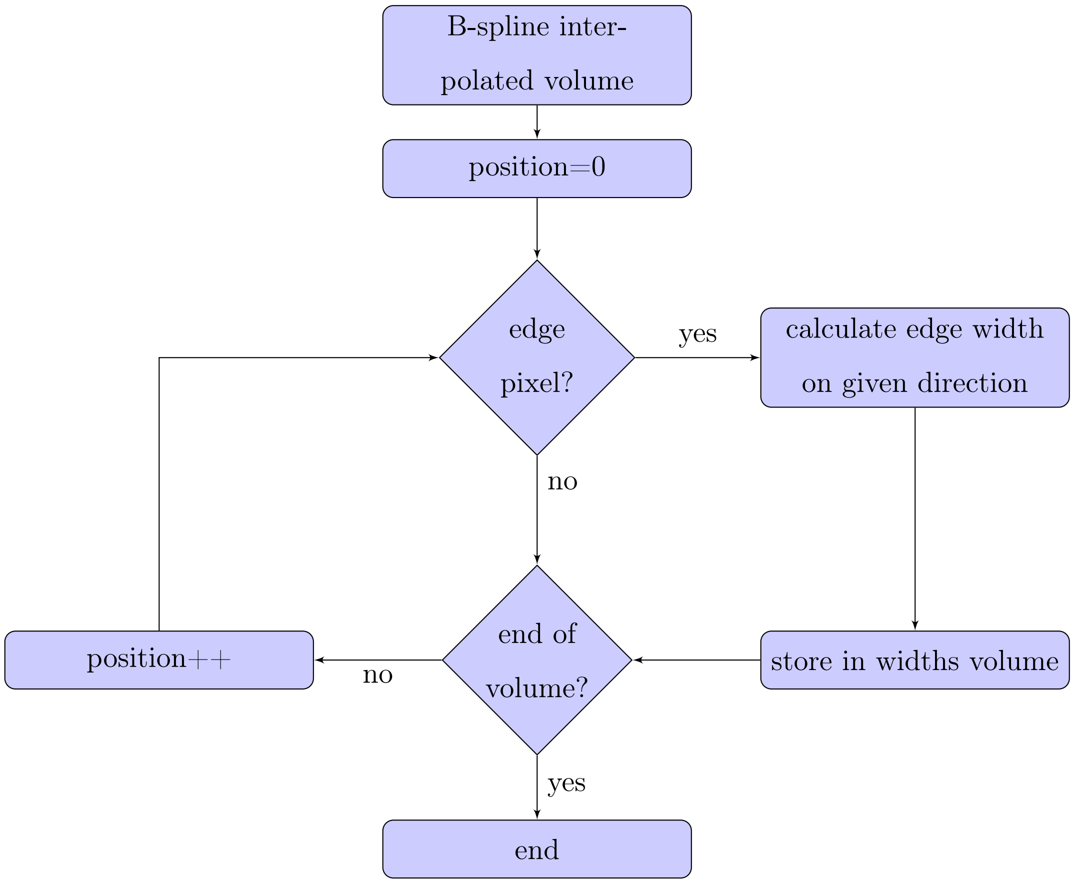


Figure 3.17: edge_width function flowchart

Once again the original input volume is used to calculate the B-spline interpolated volume. As we explained in the requirements section, this is needed to enable a much sharper edge width calculation. "edge_width" will receive 4 input parameters: B-spline coefficient volume, edge volume, a newly created widths volume (to store local widths), a vector to indicate the direction of the extrema search and a step size, which will determine how big the increment between iterations will be.

If a pixel is tagged as edge, then edge_width function will search for the closest maximum and minimum pixel value, the local width being the distance among them. Figure 3.18 shows the numerical values of the vector situated at slice 80 and column 70 (in (1,0,0) direction).

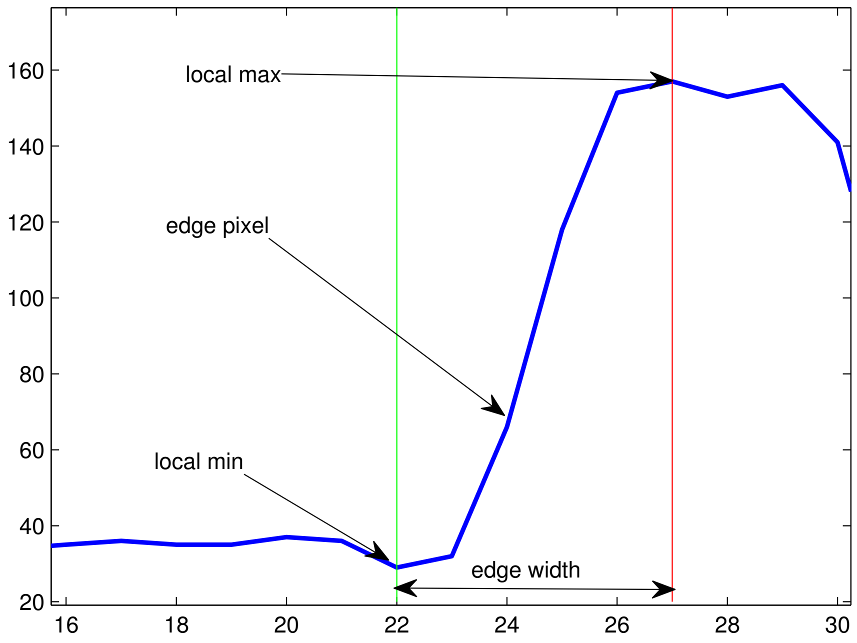


Figure 3.18: Edge width explanation

Pixel 24 is tagged as edge, at pixel 27 we find the local maximum and at pixel 22 we find the local minimum. Consequently, the local edge width associated to pixel (80,70,24) is 5 pixels when considering (1,0,0) direction. This is done for every pixel in the volume, and for every possible direction. Even though different edge widths will be found for the same pixels depending on which direction we are searching in, only the minimum width will be considered.

Partial results are displayed here as independent volumes, but this is only for illustrative purposes.

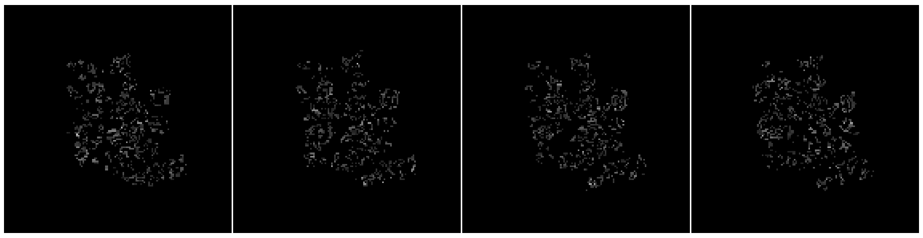


Figure 3.19: Widths in (1,0,0)

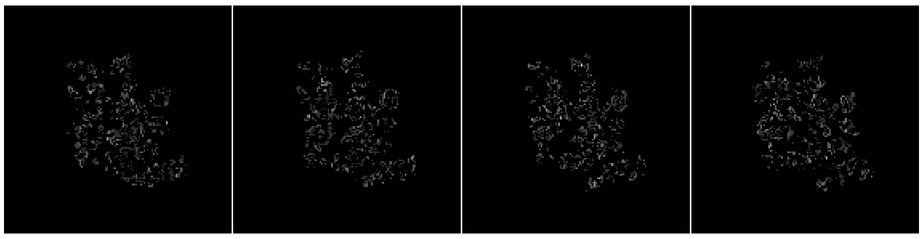


Figure 3.20: Widths in (0,1,0)

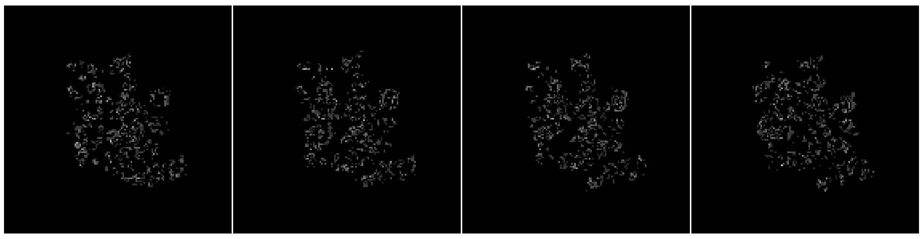


Figure 3.21: Widths in (0,0,1)

The real process consists of the function overwriting the width value stored in that position if the width found is inferior, and so, after the function is run on a new direction, the widths stored in the volume will be more precise, until eventually the final result is achieved after all directions have been examined (Figure 3.22).

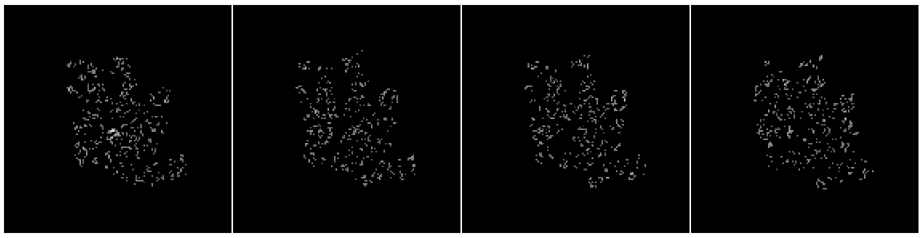


Figure 3.22: Final width volume

All that remains is to calculate the average width per edge pixel:

$$avg\ edgewidth = \frac{\sum edge\ width}{number\ of\ edge\ pixels}$$

The quality estimation metric we propose is:

$$metric = \frac{1}{avg\ edgewidth}$$

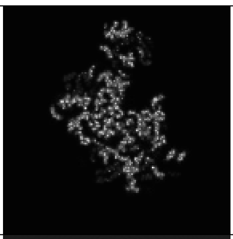
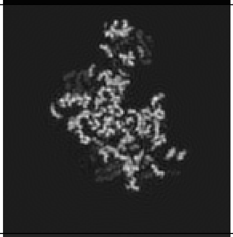
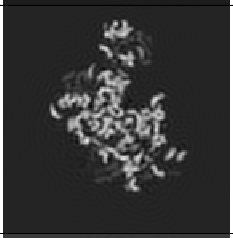
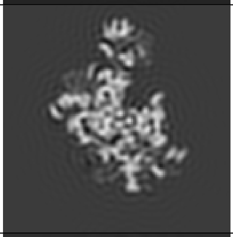
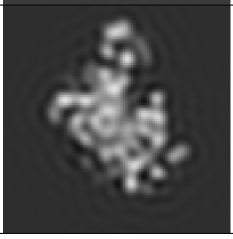
These two operations are implemented in the "calculateIBW" function.

Chapter 4

Results and analysis

The tests performed in this final stage follow the same pattern as the preceding ones in this thesis: volumes with known resolutions are used, and the program is run with these as inputs. We then compare the results obtained with the resolution of the input and attempt to find a linear relationship. Noiseless volumes are considered first, followed immediately by noisy inputs. Finally, we compare both noiseless and noisy results and draw conclusions about linearity and how noise affects the program's performance.

4.1 Noiseless performance

Sample	Resolution	Resolution IBW
	0.5	0.296782
	0.4	0.176937
	0.3	0.120723
	0.2	0.0873584
	0.1	0.0446705

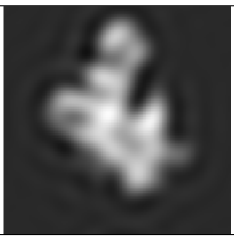
Sample	Resolution	Resolution IBW
	0.05	0.0234004

Table 4.1: Quality index estimation for noiseless volumes

It is clear that decreasing resolution implies decreasing the quality index estimation value. Figure 4.1 shows this relationship visually:

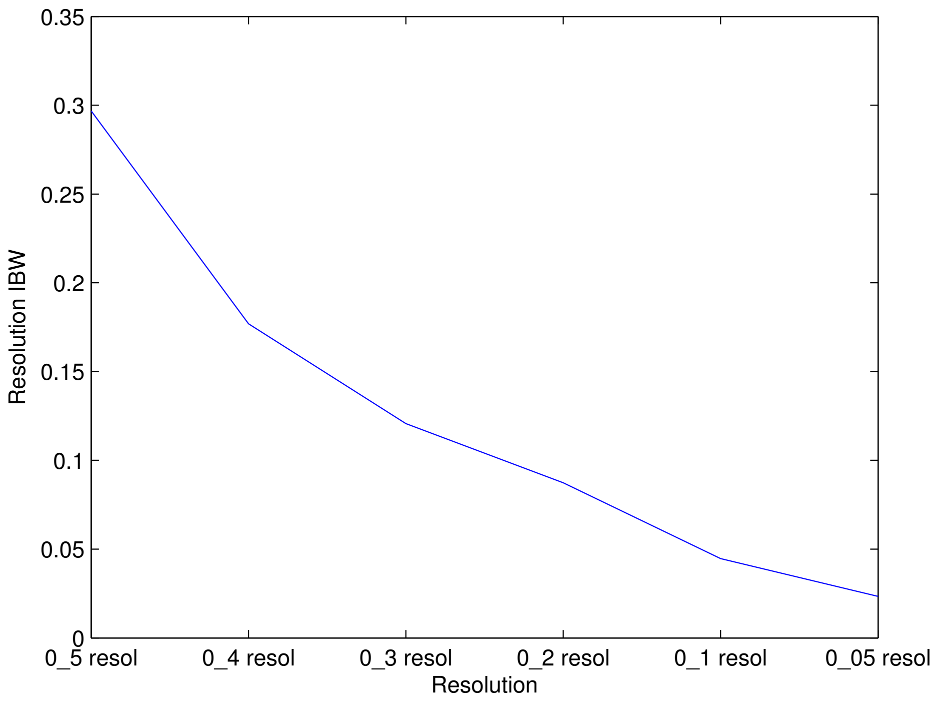
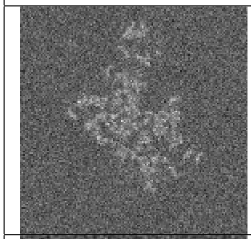
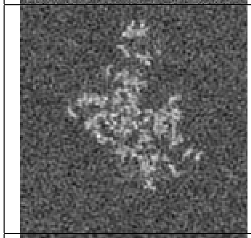
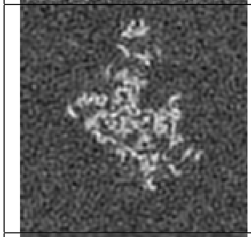
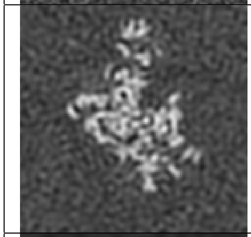
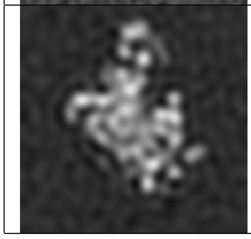


Figure 4.1: Noiseless performance

Our program behaves as expected when working with noiseless volumes, but, as real volumes have noise, further testing is still needed.

4.2 Noisy performance

Sample	Resolution	Resolution IBW
	0.5	0.511684
	0.4	0.194247
	0.3	0.128721
	0.2	0.0882885
	0.1	0.0.0451057

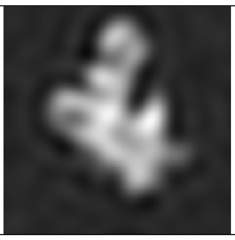
Sample	Resolution	Resolution IBW
	0.05	0.0236174

Table 4.2: Quality index estimation for noisy volumes

Figure 4.2 shows the Resolution IBW values for every resolution, and here we can discern that decreasing resolution once again implies decreasing IBW values. However, we also find that the value for 0.5 resolution breaks the linear tendency of the other values:

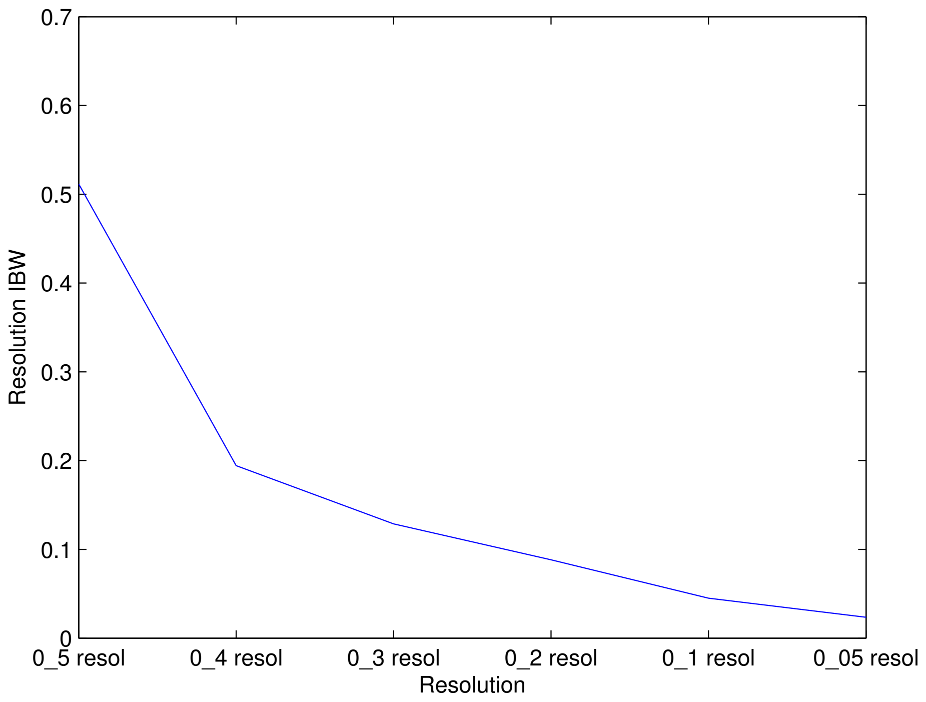


Figure 4.2: Noisy performance

4.3 Analysis

Figure 4.3 shows both noiseless and noisy performance graphs overlapped. We can see that Resolution IBW values are very similar in both environments, having in addition a linear relationship with the true resolution of the volumes.

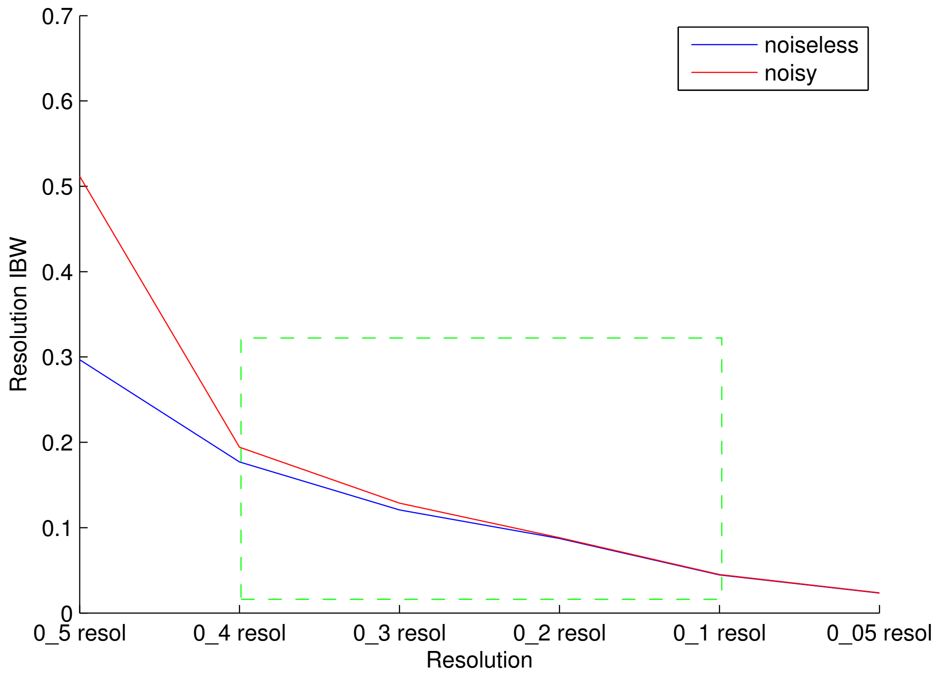


Figure 4.3: Comparison

The exception to this, however, is when 0.5 resolution noisy volume is analyzed, as Resolution IBW for that volume nearly doubles the expected value. This poses no problem for the successful use of this program, as 0.5 and 0.05 resolutions are two extreme cases which are very unlikely to occur in real experiments. The typical resolution values that an electron microscope would produce are indicated in Figure 4.3 by a green box, and it can be easily observed that for these values the program's performance is optimal.

Chapter 5

Conclusions and future steps

When, back in September, the objectives of this thesis were established, we wanted to design and implement a program which could help researchers determine the sharpness of a 3D micrograph reconstruction volume using no other reference but the volume itself.

Several papers by different researchers were studied, in which blind image quality estimation metrics were proposed for conventional images. One of these was chosen and modifications were made to ensure good performance when dealing with reconstruction volumes.

The final program meets all of the initial requirements, as 3D computing, sharp resolution and noise resistance were considered in the design phase. Test results prove that the estimation produced has a highly linear relationship with true resolution, and is also extremely robust against noise.

For future improvement, more thorough testing is proposed. We have proven the relationship between Resolution IBW and the true resolution of the volume using 12 different volumes, but, as these volumes were produced by filtering and adding noise to the original volume, it would be extremely useful if the program were tested using other micrographs and other types of molecules. This would allow a more precise understanding of how the program behaves in all situations.

Appendix

.1 resolution_ibw

```
/*↩
   **********************************************************************↩

 *
 * Authors:     Carlos Oscar S. Sorzano (coss@cnb.csic.es)
 *                      Alvaro Capell
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
 * 02111-1307  USA
 *
 *  All comments concerning this program package may be sent to the
 *  e-mail address 'xmipp@cnb.csic.es'
 ***************************************************************************↩
     */

#include <iostream>
#include "resolution_ibw.h"
#include <data/filters.h>

/* Read parameters ↩
    ------------------------------------------------------------ */
void ProgResolutionIBW::readParams()
{
    fnVol = getParam("-i");
    fnOut = getParam("-o");
}

/* Usage ↩
    ---------------------------------------------------------------------- */
void ProgResolutionIBW::defineParams()
{
    addUsageLine("Evaluate the resolution of a volume through the inverse ↩
        border widths");
    addParamsLine("   -i <file>              : Volume to evaluate");
    addParamsLine("  [-o <file=\"\">]        : Volume with the border ↩
        widths of the edge voxels");
}

/* Show ↩
    ----------------------------------------------------------------------- ↩
    */
void ProgResolutionIBW::show() const
{
      if (verbose==0)
            return;
    std::cout << "Input volume:      " << fnVol << std::endl
              << "Output widths:     " << fnOut << std::endl
    ;
}

/* Run ↩
    ------------------------------------------------------------------------ ↩
    */
//#define DEBUG
void ProgResolutionIBW::run()
{
    V.read(fnVol);

    //Mask generation
    Image<double> aux;
    double bg_mean;
    MultidimArray<double> Vmask;
    detectBackground(V(),aux(),0.1,bg_mean);
#ifdef DEBUG

    aux.write("PPPmask_no_ero_03.vol");
#endif

    //Mask volume erosion to expand the mask boundaries
    Vmask.initZeros(V());
    erode3D(aux(),Vmask, 18,0,2);

    //Correction of some flaws produced in the edges of the mask volume
    FOR_ALL_DIRECT_ELEMENTS_IN_ARRAY3D(Vmask)
    if (k<=4 || i<=4 || j<=4 ||
        k>=FINISHINGZ(Vmask)-4 || i>=FINISHINGY(Vmask)-4 || j>=FINISHINGX(↩
            Vmask)-4)
        DIRECT_A3D_ELEM(Vmask,k,i,j)=1;

    aux()=Vmask;
#ifdef DEBUG

    aux.write("PPPmask_ero_03.vol");
#endif

    //Sobel edge detection applied to original volume
    Image<double> Vedge;
    computeEdges(V(),Vedge());
#ifdef DEBUG

    Vedge.write("PPPvolume_sobel_unmask_03.vol");
#endif
```

```
    //Masked volume generation
    const MultidimArray<double> &mVedge=Vedge();
    FOR_ALL_DIRECT_ELEMENTS_IN_MULTIDIMARRAY(mVedge)
    if (DIRECT_MULTIDIM_ELEM(Vmask,n)==1)
        DIRECT_MULTIDIM_ELEM(mVedge,n)=0;
#ifdef DEBUG

    Vedge.write("volume_sobel_mask_03.vol");
#endif

    double minval, maxval, avg, stddev;

    //Invert the mask to meet computeStats_within_binary_mask requirements
    FOR_ALL_DIRECT_ELEMENTS_IN_MULTIDIMARRAY(Vmask)
    if (DIRECT_MULTIDIM_ELEM(Vmask,n)==1)
        DIRECT_MULTIDIM_ELEM(Vmask,n)=0;
    else
        DIRECT_MULTIDIM_ELEM(Vmask,n)=1;

    //Threshold is 3 times the standard deviation of unmasked pixel values
    double thresh;
    computeStats_within_binary_mask(Vmask,mVedge,minval, maxval, avg, ↩
        stddev);
    thresh=3*stddev;

    //Final edge volume generated by setting to 1 positions with values > ↩
        threshold
    Image<double> Vaux;
    Vaux().initZeros(mVedge);
    FOR_ALL_DIRECT_ELEMENTS_IN_MULTIDIMARRAY(mVedge)
    if (DIRECT_MULTIDIM_ELEM(mVedge,n)>=thresh)
        DIRECT_MULTIDIM_ELEM(Vaux(),n)=1;

#ifdef DEBUG

    Vaux.write("volumen_bordes_definitivo_03.vol");
#endif

    const MultidimArray<double> &mVaux=Vaux();
```

```
//Spline coefficient volume from original volume, to allow <1 step ↩
    sizes
MultidimArray<double> Volcoeffs;
Volcoeffs.initZeros(V());

produceSplineCoefficients(3,Volcoeffs,V());

//Width parameter volume initialization
Image<double> widths;
widths().resizeNoCopy(V());
widths().initConstant(1e5);
double step=0.25;

Matrix1D<double> direction(3);

//Calculation of edge width for 10 different directions, if a smaller ↩
    value is found for a different
//direction on a given position the former value is overwritten

//Direction (1,0,0)
VECTOR_R3(direction,1,0,0);
edgeWidth(Volcoeffs, mVaux, widths(), direction, step);

//Direction (0,1,0)
VECTOR_R3(direction,0,1,0);
edgeWidth(Volcoeffs, mVaux, widths(), direction, step);

//Direction (0,0,1)
VECTOR_R3(direction,0,0,1);
edgeWidth(Volcoeffs, mVaux, widths(), direction, step);

//Direction (1,1,0)
VECTOR_R3(direction,(1/sqrt(2)),(1/sqrt(2)),0);
edgeWidth(Volcoeffs, mVaux, widths(), direction, step);

//Direction (1,0,1)
VECTOR_R3(direction,(1/sqrt(2)),0,(1/sqrt(2)));
edgeWidth(Volcoeffs, mVaux, widths(), direction, step);

//Direction (0,1,1)
VECTOR_R3(direction,0,(1/sqrt(2)),(1/sqrt(2)));
```

```
    edgeWidth(Volcoeffs, mVaux, widths(), direction, step);

    //Direction (1,1,1)
    VECTOR_R3(direction,(1/sqrt(3)),(1/sqrt(3)),(1/sqrt(3)));
    edgeWidth(Volcoeffs, mVaux, widths(), direction, step);

    //Direction (-1,1,1)
    VECTOR_R3(direction,-(1/sqrt(3)),(1/sqrt(3)),(1/sqrt(3)));
    edgeWidth(Volcoeffs, mVaux, widths(), direction, step);

    //Direction (1,1,-1)
    VECTOR_R3(direction,(1/sqrt(3)),(1/sqrt(3)),-(1/sqrt(3)));
    edgeWidth(Volcoeffs, mVaux, widths(), direction, step);

    //Direction (1,-1,1)
    VECTOR_R3(direction,(1/sqrt(3)),-(1/sqrt(3)),(1/sqrt(3)));
    edgeWidth(Volcoeffs, mVaux, widths(), direction, step);

#ifdef DEBUG

    std::cout << "width stats: ";
    widths().printStats();
    std::cout << std::endl;
    widths.write("PPPwidths.vol");
#endif

    double ibw=calculateIBW(widths());
    std::cout << "Resolution ibw= " << ibw << std::endl;
    if (fnOut!="")
      widths.write(fnOut);
}
```

.2 edge_width

```
void ProgResolutionIBW::edgeWidth(const MultidimArray<double> &volCoeffs,
                                  const MultidimArray<double> &edges,
                                  MultidimArray <double>& widths, const ↩
                                      Matrix1D<double> &dir,
                                  double step) const
{
    double forward_count, backward_count, slope;
    Matrix1D<double> pos_aux_fw(3), pos_aux_bw(3), pos(3), pos_aux(3), ↩
        next_pos(3), Kdir;

    Kdir=step*dir;

    //Visit all elements in volume
    FOR_ALL_ELEMENTS_IN_ARRAY3D(edges)
    {
        //Check for border pixels
        if (A3D_ELEM(edges,k,i,j)!=0)
        {
            //reset all counters
            forward_count=0;
            backward_count=0;
            VECTOR_R3(pos_aux_fw,j,i,k);
            pos_aux_bw=pos=pos_aux_fw;

            //find out if pixel magnitude grows or decreases
            pos_aux=pos;
            pos_aux+=dir;
            double value_plus_dir=volCoeffs.interpolatedElementBSpline3D(↩
                XX(pos_aux),YY(pos_aux),ZZ(pos_aux));

            pos_aux=pos;
            pos_aux-=dir;
            double value_minus_dir=volCoeffs.interpolatedElementBSpline3D(↩
                XX(pos_aux),YY(pos_aux),ZZ(pos_aux));

            slope=value_plus_dir-value_minus_dir;
```

```
double sign;
if (slope>0)
    sign=1;
else
    sign=-1;

//current_pixel is multiplied by the sign, so only one ↩
    condition is enough to detect an
//extremum no matter if the pixel values increase or decrease
double current_pixel=sign*volCoeffs.↩
    interpolatedElementBSpline3D
                    (XX(pos_aux_fw),YY(pos_aux_fw),ZZ(↩
                        pos_aux_fw));

double next_pixel;
bool not_found;

//Search for local extremum ahead of the edge in the given ↩
    direction
do
{
    not_found=true;
    next_pos=pos_aux_fw+Kdir;
    next_pixel=sign*volCoeffs.interpolatedElementBSpline3D
               (XX(next_pos),YY(next_pos),ZZ(next_pos));

    if(next_pixel>current_pixel)
    {
        current_pixel=next_pixel;
        pos_aux_fw=next_pos;
        forward_count++;
    }
    else
    {
        not_found=false;
    }
}
while(not_found);

current_pixel=sign*volCoeffs.interpolatedElementBSpline3D
```

```
                    (XX(pos_aux_bw),YY(pos_aux_bw),ZZ(pos_aux_bw));

            //Search for local extremum behind of the edge in the given ↩
                direction
            do
            {
                not_found=true;
                next_pos=pos_aux_bw-Kdir;
                next_pixel=sign*volCoeffs.interpolatedElementBSpline3D
                         (XX(next_pos),YY(next_pos),ZZ(next_pos));

                if(next_pixel<current_pixel)
                {
                    current_pixel=next_pixel;
                    pos_aux_bw=next_pos;
                    backward_count++;
                }
                else
                {
                    not_found=false;
                }
            }
            while(not_found);

            //If the width found for this position is smaller than the one↩
                 stores in edges volume
            //before it is overwritten
            if ((forward_count+backward_count)<A3D_ELEM(widths,k,i,j))
            {
                A3D_ELEM(widths,k,i,j)=forward_count+backward_count;
            }
        }
    }
}
```

.3 calculate_IBW

```
double ProgResolutionIBW::calculateIBW(MultidimArray <double>& widths) ↩
    const
{
    double total, count;
    total=count=0;

    FOR_ALL_ELEMENTS_IN_ARRAY3D(widths)
    {
      double width=A3D_ELEM(widths,k,i,j);
        if (width<1e4)
        {
            total+=width;
            count++;
        }
    }
    double avg = total/count;
    return 1.0/avg;
}
```

References

[1] Z. Wang, H. R. Sheikh, A. C. Bovik [2002]. "No-reference perceptual quality assessment of JPEG compressed images". IEEE Transactions on image processing, Vol. 13, No. 1, 2004

[2] S. Gabarda and G. Cristóbal. "Quality evaluation of blurred and noisy images through local entropy histograms". Proc. SPIE 6592, 659214, 2007

[3] R. Ferzli, L. Karam. "A no-reference objective image sharpness metric based on the notion of Just Noticeable Blur (JNB)". IEEE Transactions on image processing, Vol 18, No. 4, 2009

[4] P. Marziliano, F. Dufaux, S. Winkler, T. Ebrahimi. "A no-reference perceptual blur metric". Proc. ICIP (3), pp.57-60, 2002

[5] F. Fuentes, C.O.S. Sorzano. "Restauración de volúmenes de microscopía electrónica de partículas individuales". Universidad San Pablo CEU, 2010

[6] S. Jonic, C.O.S. Sorzano. "Splines for bioimaging". Optical and Digital Image Processing: Fundamentals and Applications. Eds. G. Cristóbal, P. Schelkens, H. Thienpont. Wiley VCH

[7] J. Canny. "A computational approach to edge detection". IEEE transactions on pattern analysis and machine intelligence, Vol. 8, No.6, 1986