User Guide
for
Detonation Cell Size Measurement
using
Photoshop and Matlab

J.-P. Hébral and J. E. Shepherd

Graduate Aeronautical Laboratories
California Institute of Technology
Pasadena, CA 91125

Explosion Dynamics Laboratory Report FM00-6

Summer 2000

Introduction

Detonation cells are a characteristic feature of detonation shots, and their size is a very valuable experiment parameter to know. To reduce the uncertainty in this variable that is used to being related to manual measuring, a computational processing has been developed, based on Fourier analysis of digital pictures. The objective is to highlight the relevant frequencies present in the cellular pattern, and thus get the most encountered spatial wave length, which would turn out to be the cell size.

This guide is aimed to provide with the tools, knowledge, and advice needed to compute cell size measurements using the Photoshop procedures and Matlab programs implemented and available for free download on the related web site: **www.galcit.caltech.edu/~jphebral/cellsize.html**

For any questions, please feel free to contact Jean-Philippe HÉBRAL: jphebral@galcit.caltech.edu

Contents

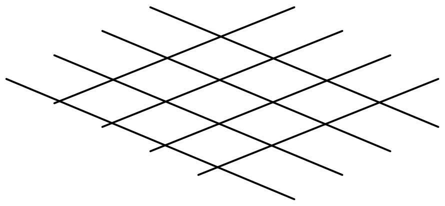
1- Theory

The purpose of this project is to calculate the average cell width of sooted foils without the high dependence upon the user sensitivity, which may result in differences ranging up to a factor of 2 between 2 measurements of the same foil.

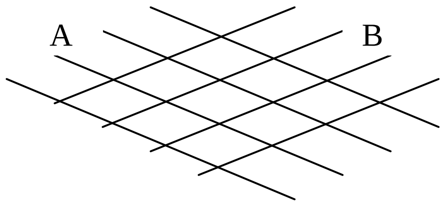
The use of computing facilities lets us obtain a result independent of the user, which is the compulsory condition to repeatable and trustful data analysis.

The method used in this context is the spectral analysis with the help of the 2-dimensional Fourier Transform.

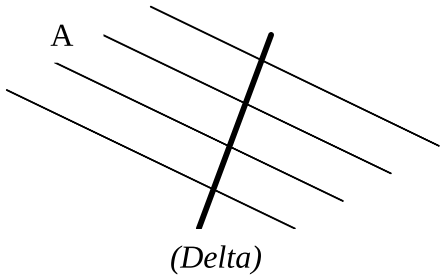
Let's assume a perfect cellular pattern:



We can consider this pattern as composed out of 2 sets of parallel lines, that we'll name A and B.

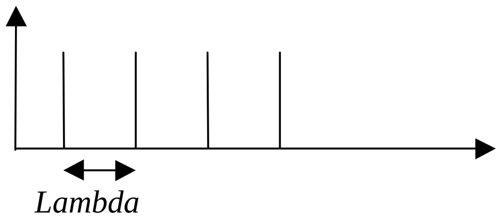


Let's consider the set A and the common perpendicular Δ to its lines.



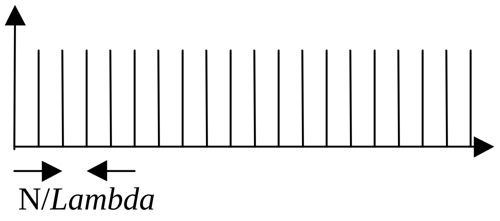
(Delta)

Suppose that you run along *(Delta)*. Each time you meet a line, the intensity of the image can be simulated by a discontinuity, compared to the uniform background intensity. Thus, the intensity along *(Delta)* is plotted as follows:



Lambda is the spatial wavelength, i.e. the cell width. Our aim is to determine *Lambda*.

The Fourier Transform of the previous function (known as a comb function) returns:

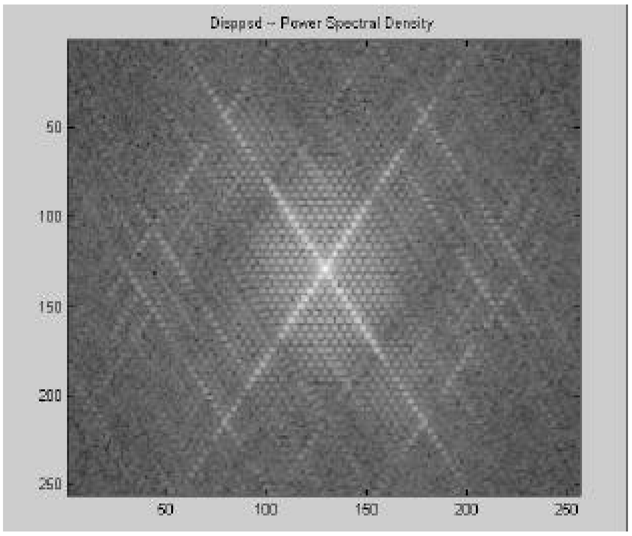


So we still get a comb function, with a characteristic frequency of N/*Lambda*, where N is the length of the sample.

This is the basic theory of what happens when we compute the Fourier Transform.

The 2D Fourier Transform computes the same way for the sets A and B. The result is the well-known image of 2 legs (which are in fact the 2 corresponding lines $(Delta)_A$ and $(Delta)_B$; thus, they can give us the angle of the cell orientation, provided that the photograph was taken straight… If the lines A make an angle *Theta* with the horizontal direction, the corresponding leg in the PSD then makes an angle of *Pi/2-Theta* with the same direction).

Those legs are basically made out of numerous dots, corresponding to the comb function obtained after the Fourier Transform.



Thus, the wider the cells are, the smaller the spacing between the dots in the PSD.

Elements in the PSD other than the 2 legs are only the result of noise. Don't put any interest into them.

In the PSD, we notice that the legs fade out, then reappear a little, then fade out again… It highlights the fact that the lines A and B of the cellular pattern have a non negligible width, which can be assimilated to a periodical signal with a quite large frequency, much larger than this of the cellular pattern. This assumption is confirmed by the PSD image, where one notice that the intensity variation caused by the dots is much quicker than the variation of the general intensity of the leg.

Now you may have some ideas of how to interpret the PSD representation.

Keep in mind that the only thing we’re looking for as of now is the spacing between the dots. Once we’ll know it, we’ll know the cell width.

2- Taking photographs

2-A- Lighting

Sorry, but the beginning is probably the most difficult thing to do right. Most of your efforts should concentrate on getting nice pictures, with as little reflection and as flat lighting as possible.

I used 2 halogen lights hung on the wall above the foil. Since they are likely to make bright trails on the foil, the light has to be diffused with a kind of plastic plate. I used one of the neon plates. The problem that then occurs is a bright lighting on the top, and darkness on the bottom. The solution is of course to light the foil from the bottom as well as from the top. I would recommend the use of 2 lights on the bottom as well, since only 1 wouldn't allow to highlight the whole width of the foil. There is still much to do in this domain, among others to find a way to fix the bottom lights (I had to hold them!). They also unfortunately don't have any diffuser in front of them.

A direction that maybe should be explored is the use of neon of the kind of those used in the lab. They provide diffuse light on a large length, and one neon on each side of the foil might give good results.

Another idea is to take the pictures outside, using the natural light. Pick a shadowy place to put the foil. One can reckon on the natural light reflection to provide a uniform lighting.

The parasite reflections are also a problem to take care of. The most obvious reflection sources are the walls and the ground, so I surrounded the photographing area with black sheets. If the photographer wears bright clothes, it can also cause a bright mark on the foil. In this case, hide behind the black sheet and press the button from there (you can also set the timer on the camera). At last, the camera itself sometimes caused impressive noise. The front face of the camera is unfortunately covered of highly reflective materials. Thus, I made a kind of cover for the camera. I also covered with narrow tape the edge of the objective (don't worry, it's harmless!).

2-B- Setting up the camera

Well, now that the photographing studio is set, let's set up the camera.

Insert the card, plug the adaptor (first on the camera, then in the wall plug) or put batteries (do not use adaptor when batteries are installed in the camera).

There are 2 photographing modes for the camera: A is fully automatic, M is customizable. I would recommend A at first. It is usually sufficient. If not, switch to M. You set the mode with the main power/trigger button.

Set the flash off by pressing once the middle button on the bottom of the camera. A distinctive icon should appear on the top right of the screen. Set the quality to Fine with the right button on the bottom. The number indicated next to Fine on the screen is the remaining amount of pictures for this memory space. In M-mode, you can get a quality named HI, higher than Fine. The image format is no more JPEG, but TIFF. Since the image is much bigger, you can only take one HI picture with the 16MB card, and the difference in quality is really not obvious. I don't think it's worth it.

The main parameter to decide if a picture is Ok or not, once the lighting is flat…, is the sharpen aspect of the image. In A-mode, the camera is supposed to auto focus. It works quite well for large, apparent cells with a high contrast with respect to the background, but may not be that satisfactory for very dark (small cells) or very bright foils. For extra small cells, the camera might be as close as 3cm from the foil, and you usually have to switch to M-mode.

- ***A-mode:***

Press the left button on the bottom until a flower appears on the screen. You can also set Flower + Timer if you wish to use the timer. The Flower indicates a close up picture (less than 2m from the subject). This will always be the case.

In A-mode, the camera focuses dynamically each time you move the objective toward a new subject (you can feel the camera slightly vibrating when it focuses. It is sometimes convenient, but if the camera doesn't manage to focus, the auto-focus will not stop and you've got 9 chances out of 10 that the image won't be focused when triggering. In this case, switch to M-mode, where you can control the focus.

If the focusing works, once you're ready, half-depress the button. When you see a continually glowing green light, the camera is ready to shoot. Then depress the button completely. It's in the box! The image might look more blurry on the screen than it is actually, so try to see what it looks like on the computer before trashing it. Of course, it is sometimes even not worth checking on the computer…

- ***M-mode:***

Useful, if not necessary, when shooting pictures for small cells. The camera might then need to be very close from the foil.

If you do not make any further setting, the M-mode is exactly similar to the A-mode. So you can still set the focus to close-up (Flower) if you want. Don't forget to turn the flash off anyway.

The M-mode is useful when the autofocus is not satisfactory. The solution is then to tell the camera at what distance you are from the subject. Do it by holding the left button on the bottom and by turning the rolling button on the top of the camera. The flower will then disappear and be replaced by a distance. Choose the one that best fits your installation. If the color of the indicated distance turns red, then this configuration Zoom + Focus distance isn't available. In this case, you have to change either the distance of the camera or the zoom (remember that you can access a digital zoom by holding the zoom-in button when the zoom bar has reached the T). Notice that when you've set a focus distance, the camera doesn't auto focus anymore if you move your camera toward another point. To turn back to autofocus, press the left bottom button.

The image may look sharp on the screen in some cases. Don't mind about that, and think that images usually look worse than they are (what else can we ask?!). So give it a try on the computer. With some experience, you may be able to say if the picture is gonna be Ok just by looking at the blurry bad-colored displayed picture on the LCD screen…

You can eventually take the picture the same way as described in the A-mode section (half-depress, green light, depress…).

2-C- Getting the pictures from the camera to the PC

Getting the pictures from the card to the computer is just as getting files from a floppy: insert the card in the reader, and open the folder:

Removable Disk (F:)/DCIM/100NIKON

You'll then find your pictures, plus an Info file containing the characteristics of each picture (you can doutless ignore this file). Just click and drag your pictures to their destination folder. You can rename them without problem and delete them from the card to save space.

The only thing to care about is the removal of the card from the reader: don't just pull on it, but select first the command:

Start menu > Programs > ImageMate > Safe Swap

Click 'Swap'. You can now safely remove your card form the reader. Swapping allows the card to make effective the changes that occurred on it when inserted in the reader (Deletion or addition of files…). It doesn't format the card, so you won't lose what is still on it.

3- Using Photoshop

Photoshop is basically a software that allows to create new images as well as to modify existing pictures. It is a very comprehensive tool, that may seem a bit confusing in the beginning, but you will be far from needing to know everything about it. Let's try to get used to it, and to know basic principles. There are only a few tools and actions that you should be familiar with.

Launch Photoshop and open any picture. Inspect the working area.

3-A- Useful tools

On the left is the tool panel. The tools you should get used to are:

- **The marquee tool**, 1st from the top, on the left: used to make selections by dragging a selection box on your picture. Hold Schift by dragging the selection to constrain the box to a square. Click in the selection and drag it to move it.
- **The crop tool**, same box as the marquee tool (click on the box to get all the available tools to appear; the crop tool is the last one). It works the same way as the Selection tool. Drag a box (holding Shift to constrain it to a square). The handles allow then to modify the shape of the box. You may also find that you can rotate, distort,… the box, but it's useless in our aim and it might confuse you for starting. Once you've got the shape you wish, press enter. If you changed your mind and don't want to crop anymore, click on any other tool and select "Don't crop".
- **The ruler**, 8th from the top on the left: used to measure distances and angles. Set the unit in File>Preferences>Units & Rulers. I recommend 'Pixels', since the cm, inches,… won't probably be actual, and you might get confused with it. Then drag a line on the picture. In the Info window on the right of the screen, you will be able to read:
  - A: the angle of the dragged line in respect with the horizontal
  - D: the length of your line
  - W & H: the width and height of your line

  You can also measure the angle of the line in respect with another direction than horizontal: once you've dragged the first line, go on one of its ends, hold Alt and drag another line. A is now the angle between both lines, and D1 and D2 the lengths of both lines.
  You may at any time constrain the dragged lines to increments of 45 degrees by holding Shift when dragging.
- **The magnifying glass**, 10th from the top, on the right: it is preset as a Zoom-In command. You can turn it into Zoom-Out by holding Alt when clicking.

3-B- Useful Palettes

On the right, the panels you may find useful are:

- **Navigator**, top right palette: allows you to select quickly the displayed area of the picture and its magnifying ratio.
- **Info**, top right palette: indicates the information of the selected tool. We just saw an example with the ruler tool. Other indications may include (depends on the selected tool) CMYK (color information), X & Y (coordinates of the starting or ending point, according to the tool), W & H (width and height of the selection box, of the horizontal and vertical skew,…)
- **History**, 3rd palette from the top: everything you do to the picture is temporally stored in the history. If you wish to cancel an action, click on the preceding icon in the History palette. You will then come back to the state preceding this action. You can still go forward and retrieve the cancelled action. But the next action you'll do after coming back in the history will erase and replace all the actions you canceled by coming back (those that are shaded in gray).

3-C- Useful Menus

The menus on the top of the screen may contain some useful features.

- **Image:**
  - Mode>grayscale turns your color picture into a Black and White image. This has to be done before using Matlab.
  - Adjust>Auto Levels and Auto Contrast (once you'll be used to it, you may use Levels and Contrast directly…) are convenient to increase the contrast and to modify the light on the picture. I am not convinced that they are necessary, since they may also highlight the stains and noisy dots of the picture, which is not wished. If you have doubts, don't use them.
- **Select:**
  - You may enjoy using the first features (All, Deselect, Reselect, Inverse) to make quick selections, cancel the last selection, reselect it if you deselected it, and take the complement of your selection with respect to the whole image.
- **Filters:**
  - A filter affects the current selection. If no selection is done, it affects the whole image. We will use Blur and Sharpen. We'll see later how.

That's it! If you can handle a little bit all those few features, you're ready to use Photoshop for cell size measurement.

4- Image Pre-processing

Launch Photoshop. Go in the File menu>Preferences>Units & rulers and check that the unit used is 'Pixels'.

4-A- Gray scaling

The first thing to do once you've launched Photoshop and opened your image is to turn it into a Black & White picture, the only kind of picture supported by the programs. Go in the Image menu, and select Mode>Grayscale. That's it!

4-B- Adjusting the image size

The whole image is not suitable for image processing. A part of it may be the ruler, and it is likely to disturb the computation. Thus, you have to select the area of the picture that you want to use. Select the Crop Tool (hold the selection tool: top left button of the Tool Panel; the crop tool is the last one), click on the Info Menu in the top right window, and drag a crop box in the picture on the interesting area. You can see the dimensions in pixels of the box in the Info Menu on the top right (W=Width, H=Height). To constraint the box to a square, hold Shift by dragging. It is highly recommended to make a square selection, and slightly recommended to take a dimension that is a power of 2 (256 or 512 are common values).

Once your selection is made, press enter to crop the picture to the selected area. Save your study area with an obvious name. For instance, if you choose to name every original picture with the number of the shot and an index for the number of the picture for this shot ('1856-2.jpg' for instance), you should consider to call the cropped image '1856-2-work.jpg' for instance. Just give names such that no mistake is possible.

An important thing to keep in mind is that the original and the cropped one must always remain in the same proportions. Do not resize the new image to make it bigger. If you fail to do so, you will get results in pixels, but you won't be able to turn them into cm, since the pixels of the image you've worked on and the pixels of the picture containing the ruler are not the same… (Notice: magnifying does not affect the image size, you can do it fearless).

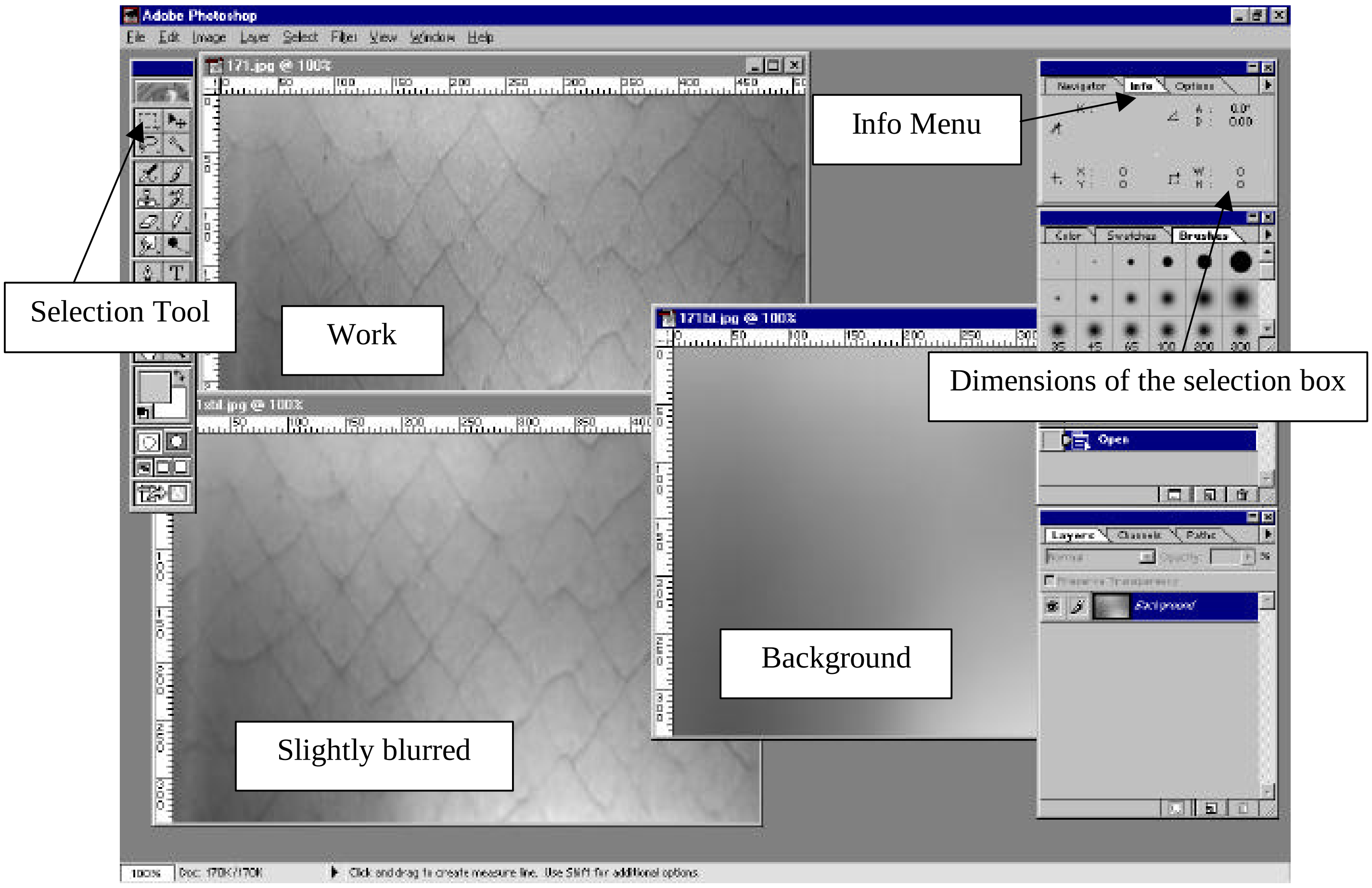
Starting now, we're only working on the cropped image.

4-C- Preprocessing

The greatest problem in taking pictures is the heterogeneity of the image. The light must be as regular as possible (avoid light and dark areas), and the noisy details as smooth as possible. Otherwise, they can completely disrupt the edge enhancement by creating a bunch of little dots, scratches, stains among the picture. This 'snowy cloud' will make barely possible to see the cell boundaries. Thus, the concern is now to get rid of the irregularities of the picture.

A tip is to slightly blur the picture to get rid of the irregular details. Then, by going on blurring, you get rid of everything on the image but the background, which contains the lighting irregularities of the picture. By subtracting the background from the slightly blurred image, you get a quite homogeneous picture, with a flat light. This is the principle.

Concretely, select Filter>Blur>Gaussian blur, and set a radius between 1 and 2 (it may depend on the picture) to get the slightly blurred image. Save it as ‘1856-2-sbl.jpg’, with the same notation as before. Now set a higher radius (10 to 12, approximately). You have to get a picture where the cells have disappeared, but the lighting is about the same as the original (if the radius is too high, the lighting is flat, which is not good). This is what we call the background. Save it as ‘1856-2-bck.jpg’.

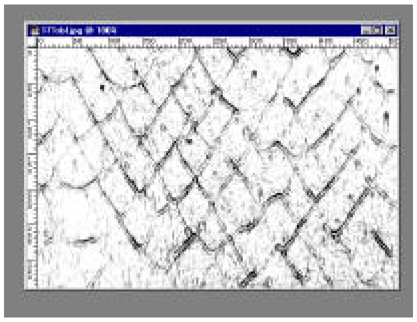


Photoshop doesn’t allow to subtract pictures together. This will be done with Matlab.

4-D- Last minute discovery – alternate preprocessing

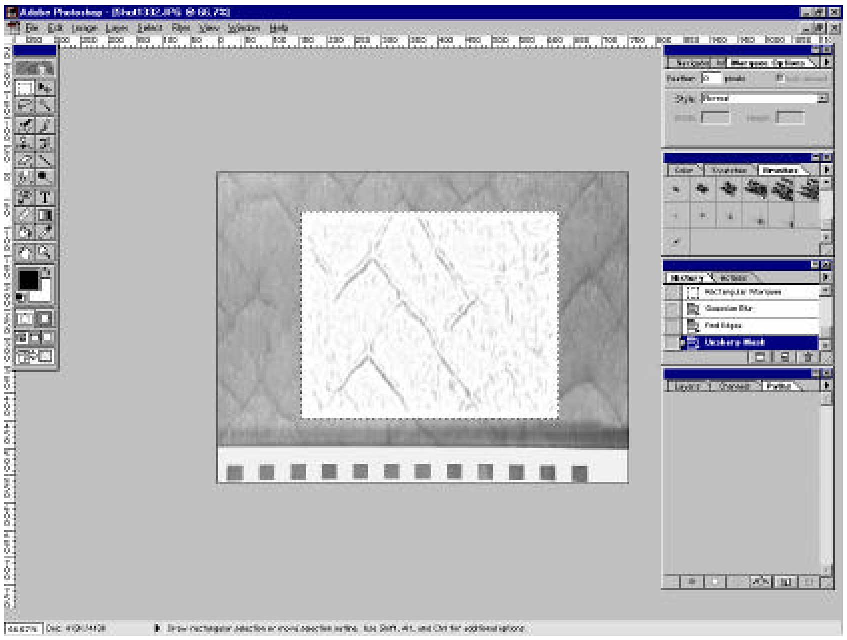
This is just another way to improve the quality of the image before running the program. It may exist tremendous of them! For a first reading, you may skip this part, since it might be a little bit confusing (I'll use stuffs that are discussed later in this report…).

Take the slightly-blurred--and--cropped image in Photoshop. Select Filter>Stylize>Find edges. Then select Filter>Sharpen>Unsharp mask, set the Amount and Radius to max and the threshold to min. Have a look at the preview before clicking 'OK'. Not too bad, isn't it? Well, mine wasn't that bad… If it's not very nice, keep Amount to max, Threshold to min, and try to change the radius.



4-E- Warning

A common error would be to try to enhance the edges on the very original picture, before you crop it. Thus, it may still contain the ruler, whose brightness is much higher than the one of the foil. It may cause problems, since the blurring, find edges, and sharpen commands use the average characteristics of the picture, which will include the ruler. You might then get weird results. I'm not sure if I am very clear… Anyway, try to only use pure-cell foils for the preprocessing. To the limit, you can use the whole image and work on a selection in the cell area, as shown in the image below.



5- Basic use of Matlab

Some stuffs to know to avoid frequent problems with Matlab…

Matlab is a scientific calculation software. It can be used to make calculation (3+2,…), to solve equations, to plot functions,…

The main window is the command window. You just type what you want to do after the '>>' prompt sign, then press enter. The command is executed, and you get a new '>>'.

The usual actions are of course available: you can affect a value to variable (x=2 for instance) or make classical calculation. You can also use pre-existing commands, or launch programs that you will have written.

Programs are edited in a separate window. Their extension is '.m'. To edit or create a new program, just click the 'Open' or 'New' icons on the top of the main window. The cell size measurement uses programs that were written for this purpose.

Your best friend in Matlab is the command **help**. Just type 'help name_of_command', and you'll get some information on how to use this command, its use,… The help command is also useful to check if a name is already used for a pre-existing command. Check it out before naming your freshly created program, or you could overwrite an existing command of Matlab (until you close the session, it's not forever!).

Matlab is case-sensitive. The variable 'MATRIX' is not the same as the variable 'matrix'.

If you type a calculation without storing the result in a variable, the result is temporally stored in the default variable 'ans', until the next similar calculation which replaces the value stored in 'ans'. Good to know if you launch a long program, and you forgot to affect the result into a variable name. In this case, you don't have to launch the program again, just go and get the result in 'ans'.

When you compute calculation on large matrixes, you may want to avoid the display of the result (A*B when A and B are 1000*1000 is not very interesting to see…). You can forbid the display of the result of a command by adding the sign ';' after the command. For instance:

```
>> A*B        displays the result,
>> A*B;       does not.
```

The basic thing about Matlab is *Everything is a matrix*. That's why matrixes are so important in the programs, and why you have to be careful to use the semi-colon in computations that use matrixes. In case you forget it and you start seeing thousands of matrix entries running on the screen, the **Stop-command** for any computation in general is **Ctrl-C**. Also useful when you forgot to make an exit condition in a program, and the execution never stops.

To work on pictures, Matlab first needs to transform them into matrixes. To transform the picture ‘cell.jpg’ into a matrix called ‘M’, you can use the command:

```
>> M=imread(‘cell.jpg’);
```

or rather:

```
>> M=double(imread(‘cell.jpg’));
```

‘double’ only changes the type of matrix you get (format command), and makes it able to be used in further computation. Don’t mind about it, just always use the second command.

The way Matlab turns B&W images into matrixes is the following: each matrix entry corresponds to a pixel in the picture. Thus, the number of rows in the matrix is the number of pixels in the vertical direction of the image. Same correspondence between lines and pixels horizontally. Then, for each pixel of the picture, Matlab affects a value to the corresponding entry in the matrix. The values range from 0 to 255 and stands for the intensity of the pixel. 0=Black, 255=White. Between 0 and 255, a linear variation of grays.

To save a picture generated by Matlab, the command to use is:

```
>> print(#offigure, ‘-djpeg’, ‘nameoffile’);
```

This writes the figure number *#offigure* as a file *nameoffile.jpg*. This file is created in the folder C:/Matlabr11/work. To save the current active figure, type *gcf* instead of *#offigure*.

Different formats are available, among which PS, EPS, TIFF, JPEG. To select the format, type *–dps*, *-deps*, *-dtiff*,…instead of *–djpeg* in the example above.

6- Starting to use the programs

6-A- Files, folders and paths

The first thing to do is to gather all the files you're going to need in a couple of folders. You can create a folder for the pictures, and a folder for the programs for instance. Let's say that you rather put all your files in the same folder, that you named it 'works' and that it's located in the root directory of your Unix account. Now you have to add this folder to the set of folders that Matlab scans when it looks for called programs, files, variables,… To do it, type:

```
>> addpath Z:/your_user_name/works
```

Repeat this command for every folder that you're going to need. You can check that the addition is effective by typing

```
>> path
```

and seeing if your folders have been successfully added to the Matlab path.

OK, now you're ready to launch the programs that will give you the ultimate cell size!

6-B- How to handle the programs

In September 2000, there were 11 programs used in the cell size measurement. The main entry to this procedure is the program 'cellsizer', that provides a user-friendly interface to go through the computation. This program uses all the other ones.

You may also sometimes want to only use one of the programs separately. This is possible. If you don't really know the syntax to call the programs, or if you're not sure about the number and type of arguments you have to provide for each program, just use the help (type >> *help name_of_the_program*).

Some programs allow different kinds of argument. Most of them accept either the name of the picture ('cell.jpg' for instance) as an argument, or the matrix of this picture (the matrix M calculated in section 5).

If you use a program separately from others, you may want to call it using the name of the picture, which is usually more convenient. But when the program is called within a larger program, it may be wise to call it using the matrix of the picture if it was calculated before. Since this matrix is always needed to compute calculation on the image, you'll save computing time by giving it as an argument when you already have calculated it.

Anyway, when you're not sure what kind of argument is expected with this command, just use help. There's no shame in it, and it's usually quick and useful!

Now let's start with the main program: cellsizer

You can do everything just only with it (as far as cell size measurement is concerned…).

It accepts the name of the image or the matrix of the image as an argument. Check the help (>> help cellsizer) for exact examples of syntax. The image must nonetheless fulfill specific conditions:

- It must be black and white.
- It must be square (otherwise, some options may not be available, and some results may be wrong…)
- Its number of rows/columns should preferably be a power of 2 (256 or 512, larger may make the computation very slow…)

Even if you can truncate the matrix in Matlab, it is better to do it in Photoshop, since you may then be able to crop uninteresting parts and to keep the relevant ones. You can hardly visualize it in Matlab.

So I'll suppose that you followed the instructions in the Photoshop section of this report. You have now several pictures coming from the same original. We will only need the slightly blurred and the background ones, supposedly named '1856-2-sbl.jpg' and '1856-2-bck.jpg'. Let's turn them into matrixes. Do you remember the command?

```
>> Msbl=double(imread('1856-2-sbl.jpg'));
>> Mbck=double(imread('1856-2-bck.jpg'));
```

And now, we can substract the background from the blurred image:

```
>> M=Msbl-Mbck;
```

Don't forget the semi-colon! Oh, yes, forget it and see what happens! Then you'll never forget it again…Remember, you can stop any calculation with Ctrl-C.

Now the matrix M is ready to be input as an argument for any processing.

Once you've got the right picture or matrix, go on!

7- The usual procedure

For the first time, you may enjoy the use of the synthetic image ‘cell.jpg’. It shows how the program works with a perfect cellular pattern. *If you want to try with the picture you’ve just processed by blurring and removing the background, type*

>> cellsizer(M);

Let’s have a look at what happens when you type

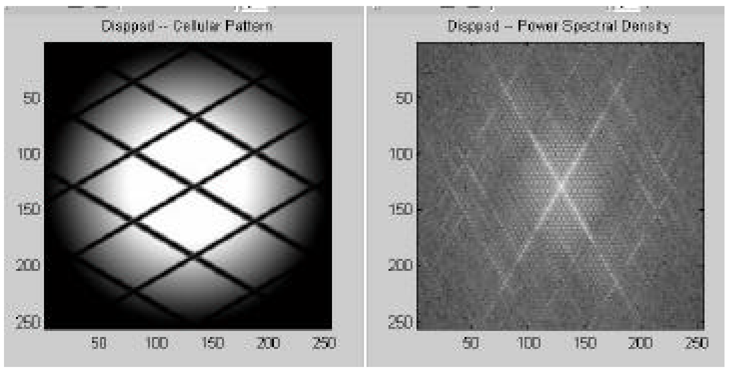
>> cellsizer(‘cell.jpg’); for instance.

The main menu is displayed on the top left side of your screen.

The steps I’ll now recommend to follow may sometimes be modified, but it’s not recommended before you really know how those programs work and interact each other.

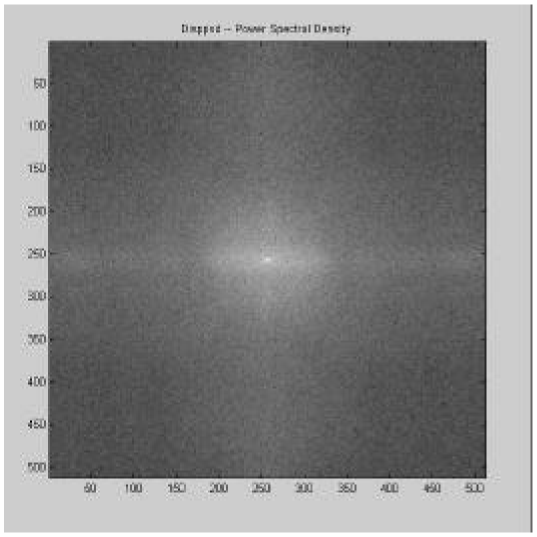
7-A- Displaying the PSD

The first thing to do is to display the cellular pattern and the PSD representation. The PSD has been automatically computed when you launched the program, but it is displayed only upon request. You don’t have to display it, it’s more for convenience than for necessity, but you may find it useful to compare the result of some image processing with the original image.



7-B- Filtering

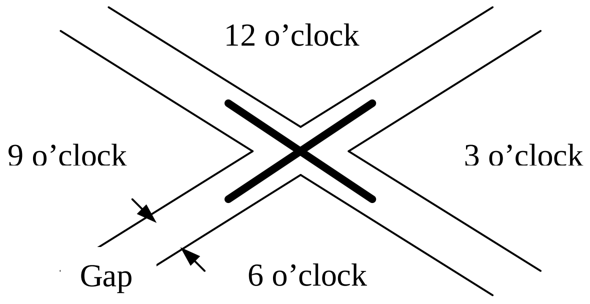
The noise filtering is not necessary for the synthetic image ‘cell.jpg’. You can process a filtering of your image if you notice some noisy frequency in the PSD. Most of the noise is usually produced by the flow stream, which will print hundreds of little



Noisy PSD;
Can you see the tiny X in the center?

parallel lines on the foil. When they are too visible, the PSD focuses on those lines, and somewhat forgets the interesting lines, which are the cell boundaries.

If so, you'll notice vertical and horizontal bright lines on the PSD. The filtering can get rid of it by erasing the unwanted areas located in the 3 and 9 o'clock or 6 and 12 o'clock area, or both. You need to enter the angles of the legs, the area you want to erase, and the gap you want to preserve around the legs.

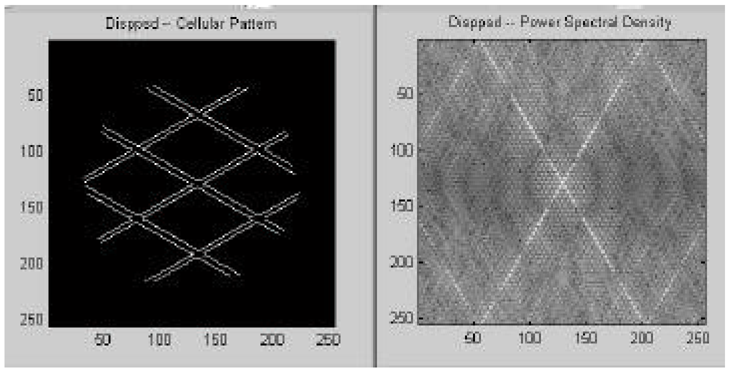


This is presented the basic principle of the filtering program. It doesn't actually work very efficiently, and efforts should be made to take pictures that wouldn't required such processing.

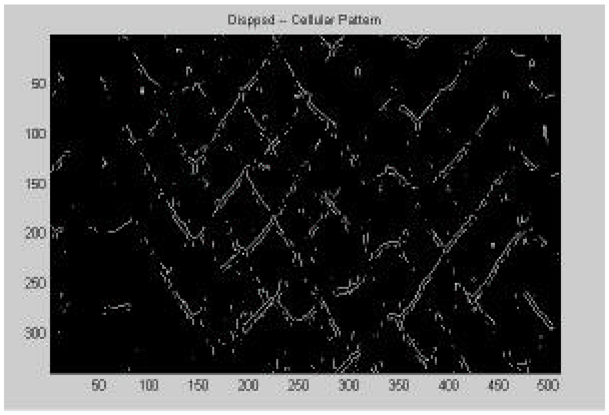
7-C- Edge enhancement

The next step is the edge enhancement. You've got the choice of using either a Matlab procedure or a home-made program. Both use the Sobel algorithm, that calculates the gradient of the image and compares it to a threshold. On each pixel, if the gradient is higher than the threshold, then it means that you consider the intensity variation as relevant. The conclusion is 'there is an edge', and the pixel turns white. In the opposite case, it turns black. Thus, you may have noticed that the smaller your threshold, the less constraining you are, and the more lines as well as stains you will get.

Both programs give you the possibility to change the threshold, to make unlimited tries, and to keep or not the obtained image.



Edge enhancement and associated new PSD for the synthetic image



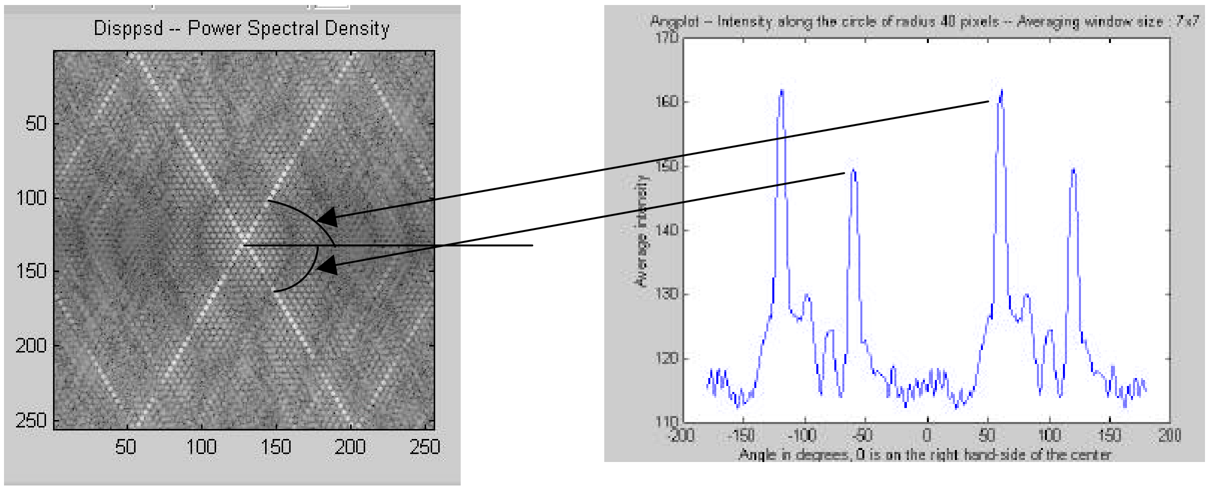
Edge enhancement for a regular picture

Your image should now show the cell boundaries in white on black. It is ready for the next processing. At this point, you may wish to display the new cellular pattern and the new PSD. It'll be useful in the next steps.

7-D- Angular scanning

To go further, we need to know the angles of the legs of the X-pattern in the PSD. To get them, we'll process an angular scanning along a circle at a specified radius, around the crossing of the legs. You should set the averaging window size to a quite high value (inputting 3 or 4 creates a 7 or 9 pixels wide window, which is usually satisfactory). The radius should be determined by using the PSD representation. Make sure that the radius will allow you to cross the legs where they are visible.

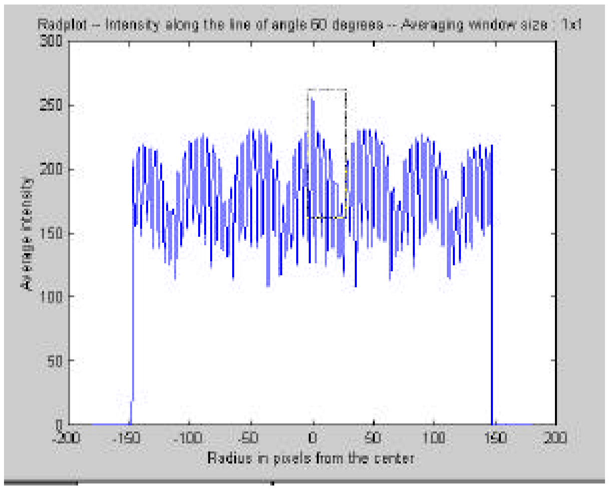
You get thus a plot of the intensity along the circle. You should see 4 apparent peaks, corresponding to each half-leg met during the scanning. If you get too much noise (plenty of acute peaks), try again with a larger averaging window (the larger the window, the smoother the plot…but the longer the computation!). If the plot is flat and you don't get any peak, your radius is too large and you're scanning too far from the crossing of the legs.



OK, now you know the angles of the legs (drag a zoom box on the top of each peak to get an accurate value). Write them down, they'll be used in the next step.

7-E- Radial scanning

We are now going to process a radial scanning along one of the legs. Select 'Radial scanning' and enter an averaging window half-size. In this case, we're trying to visualize the little dots that compose the leg. Thus, a too large smoothing must be avoided, so that typing 0 or 1 to the largest is a good starting. Enter then the angle of the leg that you want to scan.

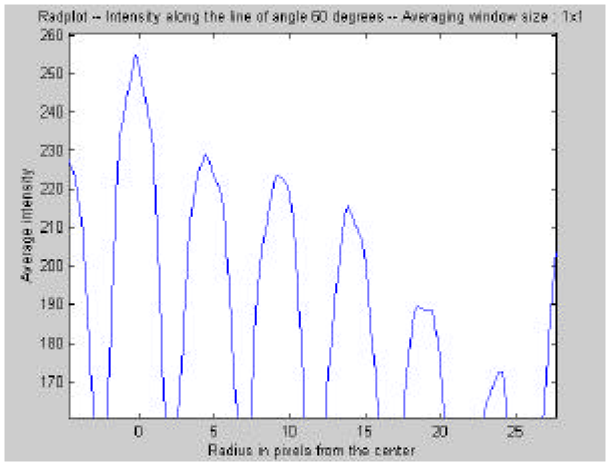


You get a plot of the intensity along this leg. The more detailed it is, the best. Zoom in (drag a box) on the right side of the DC component. Do you see regularly spaced peaks? Great, you're doing fine. Otherwise, try a smaller averaging window.

7-F- Final calculation

Those peaks correspond to the dots of the leg. The last step is to calculate the cell size. To do so, select 'Cell width'. Enter the abscissas coordinates of the 4 or 5 first peaks (but <u>do not use the DC peak</u>!). A good way to determine them is to drag a zoom box right on the top of each peak. The general syntax for the input argument is:

[abscissa1 ordinate1;abscissa2 ordinate2;abscissa3…]



In this example, your input should look like this (check on the image on the right):

[5 230;9.5 225;14 220;18 193]

When pressing enter, Matlab calculates the horizontal cell size and its harmonics (in pixels), which are the image dimension divided by the abscissas of the peaks, projected onto the horizontal axis. You may check on the cellular pattern figure (remember I said you'd need it?).

If it looks OK, you may process to the ultimate step, which is the conversion from pixel to cm unit. Use the Photoshop ruler to do it: check how many pixels a defined length in cm does (for instance 1cm), and input this correspondence in the prompt.

The final result is an array containing as a first column the cell size in cm and its harmonics, and as a second column the relative energy associated to the harmonic.

7-G- Data analysis

A convenient way to analyze the collected data is to build ‘bubble chart’ in Excel. The abscissa would be the equivalent ratio. For each value of it, you could trace some circles: the ordinate of the center would be the cell size (or the harmonic…), and the radius would be the associated energy. Thus, you would get a visual way to decide what dimension is the most significant at a given equivalent ratio.

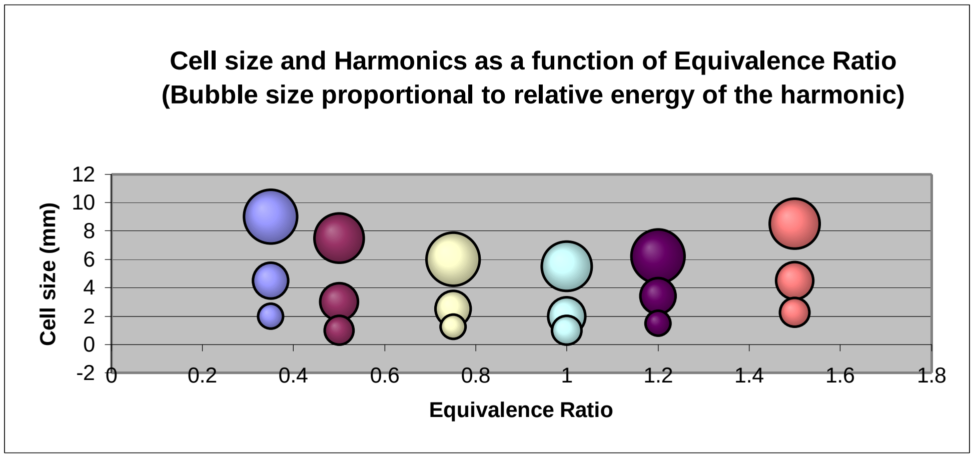
Doing so supposes that you now have a couple of data (3-4 harmonic sizes and associated energy) for some equivalent ratios.

Input those data into an Excel spreadsheet. This should look like that (the values displayed in this example are purely arbitrary):

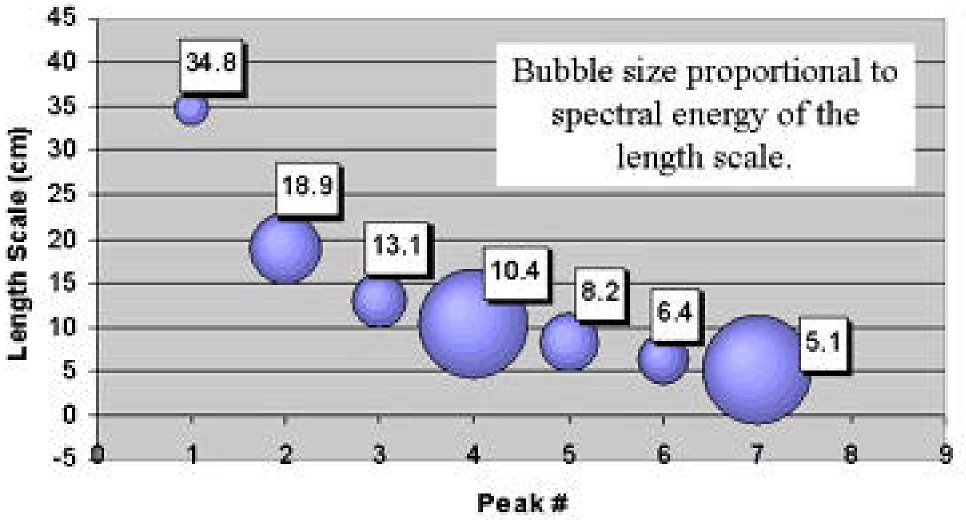
Equivalent ratio	Cell size	Energy
0.35	9	60
0.35	4.5	27
0.35	2	13
0.5	7.5	52
0.5	3	30
0.5	1	18
0.75	6	60
0.75	2.5	27
0.75	1.25	13
1	5.5	52
1	2	30
1	1	18
1.2	6.2	60
1.2	3.4	27
1.2	1.5	13
1.5	8.5	52
1.5	4.5	30
1.5	2.25	18

In most cases, one equivalence ratio will correspond to only one analyzed foil (unless you compute some averaging on data coming from different pictures of the same foil, for instance, in which case you’ll have different pictures for one equivalence ratio).

Then select this array, pick a ‘bubble chart’ (the second one adds some artistic effect on the bubbles, but both chart types are similar). Check ‘series in column’. Customize the legend, labels,… and complete the chart creation.



Enclosed below is another way of presenting the results for a given equivalence ratio (0.5 in this case). These results were obtained after sooted foil pictures from J. E. Shepherd. This series of shots was performed in 1985 at Sandia National Laboratories, using a Hydrogen-Air mixture with equivalence ratio of 0.5.



Note the multiplicity in the ordinates of the series of peaks (#1,#2,#5), (#3,#6), (#4,#7), highlighting the presence of some characteristic spatial wave lengths and their associated harmonics.

Recall that the harmonics of a fundamental frequency f *are the frequencies 2*f, *3*f, *4*f,... *Therefore, the "harmonics" of a "fundamental" wave length* L *are the wave lengths* L*/2,* L*/3,* L*/4,...*

In this case, the predominant wave length fundamental seems to be **10.4cm**. We will consider it as the dominant cell size of this foil. The presence of the other frequencies will induce fluctuations about this "most probable" value.

This result fits within a margin of 10% the ones obtained through other means (manual measurement and statistical analysis for instance).

Note the multiplicity of some ordinates, indicating some characteristic length scales and the associated harmonics.

8- The files

8-A- The images

The whole set of programs is currently stored in the Galcit directory

Z:/jphebral/summerwork/cellsizing

and on the web page

www.galcit.caltech.edu/~jphebral/cellsize.html

The Galcit directory contains not only the programs, but also a set of images that can be used to test the programs. If you want to use to those files but you don't have access to the Galcit network, please send me an-email (jphebral@galcit.caltech.edu) and I'll send you the requested files.

Here is an overview of those images:

- The folder 'grids' contains different grid samples, that can be rotated to simulate a cellular pattern. For instance, *smallhorivert* is a small grid with both horizontal and vertical lines.
- *ut* and *comics* can be used to test the edge enhancement.
- *cell* is a perfect synthetic cellular pattern. The best way to start to use the programs in Matlab. *noisycell* is the same, with some gaussian noise.
- The folder 'history' contains the images of every step of the processing applied to the same picture from the start to the end. You can refer to it when learning how to use the programs, to see how good your results are (<u>note</u>: there's no doubt that it is possible to get better results than those presented here, and the author of these lines does of course not pretend to be the ultimate reference).
    - *1original* is the picture after turning it into Black & White.
    - *2cropped* is the picture, after cropping to a square of 256*256 pix.
    - *3slightblur*, and *4bckground* are the different steps of the edge enhancement pre-processing in Photoshop.
    - *5subtracted* is the result of the operation *3slightblur-4bckground* in Matlab.
    - *6edges* is the result of the edge enhancement applied to *4subtracted*. The program *matlabsobel* was used as an example, with a threshold of 13, but you may as well use *persobel*, and vary the value of the threshold.
    - *7psd* is the PSD visualization of *6edges*.
    - *8angular* is the angular scanning, displaying 4 visible expected peaks.
    - *9radial* is the radial scanning along the leg of angle –35 degrees.
- *smallcell* is an actual photograph.

Remember that adding the path *Z:/jphebral/summerwork/cellsizing* is not enough to use the pictures contained in *Z:/jphebral/summerwork/cellsizing/grids* for instance.

You have to add the path to every subdirectory that you may want to use, i.e. *Z:/jphebral/summerwork/cellsizing/grids* if you want to use the grid images.

8-B- The programs

Let me include here the exact programs that were written during the summer 2000. The paper format may help browse through them and understand how they do work. The display and indents may not be as nice as they ought to be. A better display can be achieved by opening the files in the Matlab program editor (File>open, in the Matlab command window).

A %-sign indicates commentaries.

Here are some of the notations and programming rules I tried to respect:

- matrix names are in CAPS.
- MATRIX is the matrix of the image I'm working on.
- FINAL is the matrix of the PSD.
- HEIGHT and WIDTH deal with the size of any of those matrixes (they have the same size).
- Xc and Yc are the coordinates of the center of the PSD.
- m and sm are the variables linked to the menu and submenu present in some of the programs.
- The display of a picture still has some mystery to me. Sometimes imshow works, sometimes image, sometimes neither of those alone but together,…

cellsizer.m

```
function RESULT=cellsizer(MATRIX)

% Enters a serie of procedures to process
% the cellular pattern analysis.
%
% You can display the cellular pattern, visualize the PSD,
% launch an averaged angular scanning, which will help you
% defining the angle of the PSD legs, and use it in an
% averaged radial scanning to extract the spatial wave length
% of the cellular pattern.
%
% This programs returns an array:
%           1st column: cell size harmonics
%           2nd column: relative energy of each harmonic
%
% The picture must be Black & White.
%
% IT IS ALSO STRONGLY RECOMMENDED, IF NOT REQUIRED, TO USE SQUARE
IMAGES.
% NOT DOING SO CAN LEAD TO FALSE RESULTS...
%
% If possible, use an image whose dimensions in pixels are
% a power of 2.
%
% EXAMPLES:
%
% >> cellsizer('cell.jpg');
%    where cell.jpg is the cellular pattern image
% OR
% >> cellsizer(MATRIX);
%    where MATRIX is the double-type matrix of
%    the cellular pattern image, i.e.
MATRIX=double(imread('cell.jpg')).
%
% To keep the harmonics & associated energies in the variable ARRAY,
type
% >> ARRAY=cellsizer(...);


if isstr(MATRIX)
   MATRIX=double(imread(MATRIX)); % Turns the image into a matrix
end

pi=3.14159265;

[HEIGHT WIDTH]=size(MATRIX);
if HEIGHT~=WIDTH
   disp(' ')
   disp(' WARNING: Your image isn''t square.');
   disp(' You may love taking risks, or just getting wrong
results...');
end

FINAL=calcpsd(MATRIX,HEIGHT,WIDTH); % DSP calculation
```

```
[Xc Yc]=center(FINAL);

m=0; % Default value for the main menu

while m~=7 % If m==7 : QUIT

   m=menu('WHAT WOULD YOU LIKE TO DO?','DISPLAY CELLULAR PATTERN AND
PSD REPRESENTATION','FILTERING','EDGE ENHANCEMENT','ANGULAR
SCANNING','RADIAL SCANNING','CELL WIDTH','QUIT and learn why I wrote
this program...');

   if m==1
      disppsd(MATRIX,FINAL); % DSP visualization
   end

   if m==2
      FILTERED=filtr(MATRIX,HEIGHT,WIDTH,Xc,Yc); % Image filtering
      sm=menu('Would you like to work on the filtered image as of
now?','Yes','No');
      if sm==1
         MATRIX=double(FILTERED);
         FINAL=calcpsd(MATRIX,HEIGHT,WIDTH); % DSP calculation for the
filtered image
      end
   end

   if m==3 % Edge enhancement

      sm=menu('Which program do you wish to use?','Matlab
program','Home-made program');
      if sm==1, EDGED=matlabsobel(MATRIX);
      else, EDGED=persobel(MATRIX); end

      ssm=menu('Would you like to work on the edge enhanced image as of
now?','Yes','No');
      if ssm==1
         MATRIX=double(EDGED);
         FINAL=calcpsd(MATRIX,HEIGHT,WIDTH); % DSP calculation for the
filtered image
      end

   end

   if m==4
      angplot(FINAL,HEIGHT,WIDTH,Xc,Yc); % Angular scanning
   end

   if m==5
      ANGL=radplot(FINAL,HEIGHT,WIDTH,Xc,Yc); % Radial scanning.
Returns the angle of the scanned leg, to use it in peak.
   end

   if m==6
      if HEIGHT==WIDTH
         RESULT=peak(HEIGHT,ANGL); % Cell width calculation
      else
```

```
         disp(' Impossible to measure the cell size of a non-square
image')
      end
   end

end

disp(' ')
disp(' Thanks for running me!')
disp(' Hope to see you again soon...')
disp(' ')
disp('In fact, the reason why I wrote this program is')
why
disp(' ')
```

center.m

```
function [Xc,Yc]=center(M)

% >> [Xc,Yc]=center(MATRIX) returns as Xc and Yc the row
% and the column of the pixel of highest
% intensity in the matrix MATRIX.
%
% Usefull to find the center of a PSD image!

[m,I]=max(M);
[mm,Yc]=max(m);
Xc=I(Yc);
```

grayscale.m

```
function GRAYSCALED=grayscale(MATRIX)

% Returns a matrix whose entries have been lineary
% stretched between the initial minimum and maximum
% entries, in order to occupy the whole gray scale,
% from 0 to 255.
%
% EXAMPLE:
%
% >> GRAYSCALED=grayscale(MATRIX)


M=max(max(MATRIX));
m=min(min(MATRIX));

GRAYSCALED=255/(M-m)*(MATRIX-m);
```

calcpsd.m

```
function FINAL=calcpsd(MATRIX,HEIGHT,WIDTH);

% Computes the DSP calculation and returns the matrix of the DSP.
%
% EXAMPLES:
%
% >> calcpsd('cell.jpg')
%    where cell.jpg is the cellular pattern image
% OR
% >> calcpsd(MATRIX)
%    where MATRIX is the double-type matrix of
%    the cellular pattern image, i.e.
MATRIX=double(imread('cell.jpg')).
%
% You can also specify the dimensions of the image/matrix
% (they are the same), but it's not compulsory.
% In this case, type:
% >> calcpsd(...,HEIGHT,WIDTH)
%    where HEIGHT and WIDTH are the number of rows and columns.
%
% If you want to get the PSD matrix and call it, say, FINAL, type:
% >> FINAL=calcpsd(...);
%
% Don't forget the semi-colon, or the whole matrix will be displayed!


if isstr(MATRIX)
   MATRIX=double(imread(MATRIX)); % Turns the image into a matrix
end
% The previous loop allows us to input the image either as a function
argument
% (then CELLIMAGE is a string, for instance cell.jpg),
% or the image matrix (then CELLIMAGE is a matrix).

if nargin==1 % In case the dimensions of the matrix were not input as
arguments.
   [HEIGHT WIDTH]=size(MATRIX);
end

if nargin~=1 & nargin~=3
   error (' Number of expected input arguments: 1 or 3');
end


WIND=MATRIX.*fspecial('gaussian',[HEIGHT WIDTH],0.2*min(HEIGHT,WIDTH));
% Windows the image with a gaussian filter
WIND=grayscale(WIND); % Rescales the entries between 0 and 255.

FOURTRANS=fft2(WIND); % Calculates the 2D Fourier transform
ABSOLU=abs(FOURTRANS); % Calculates the modulus of the Fourier
coefficients
POWSPECDEN=ABSOLU.^2;
LOGA=log(POWSPECDEN);
```

```
FINAL=fftshift(LOGA); % Brings the main frequency peak in the center od
the image
FINAL=grayscale(FINAL);
```

disppsd.m

```
function disppsd(MATRIX,FINAL)

% Displays the cellular pattern and the PSD.
%
% EXAMPLES:
%
% >> disppsd('cell.jpg')
%    where cell.jpg is the cellular pattern image.
% OR
% >> disppsd(MATRIX)
%    where MATRIX is the double-type matrix of
%    the cellular pattern image, i.e.
MATRIX=double(imread('cell.jpg'));
% OR
% >> disppsd(MATRIX,FINAL)
%    where MATRIX is the same as above and FINAL is the result of
%    the PSD computation, i.e. FINAL=grayscale(calcpsd(MATRIX));


if isstr(MATRIX)
   MATRIX=double(imread(MATRIX)); % Turns the image into a matrix
end
% The previous loop allows us to input the image either as a function
argument
% (then CELLIMAGE is a string, for instance cell.jpg),
% or the image matrix (then CELLIMAGE is a matrix).

if nargin==1
   FINAL=grayscale(calcpsd(MATRIX));
end

figure;colormap(gray);imshow(grayscale(MATRIX));image(grayscale(MATRIX)
);title('Disppsd -- Cellular Pattern');axis on;zoom on; % Imshow>>Image
gives better results than Image alone. I don't know why...
figure;colormap(gray);imshow(FINAL);image(FINAL);title('Disppsd --
Power Spectral Density');axis on;zoom on;
```

filtr.m

```
function FILTERED=filtr(MATRIX,HEIGHT,WIDTH,Xc,Yc)

% Erases selected areas of the PSD.
%
% Considering the PSD visualization and its 2 two caracteristics legs,
% you can define 4 areas : 3, 6, 9, 12 o'clock.
%
% filtr allows you to erase 1 or 2 pairs of opposite areas,
% staying remote of a distance gap/2 (that you are invited to input)
% from the legs you want to keep intact.
%
% EXAMPLES:
%
% >> filtr('cell.jpg')
%    where cell.jpg is the cellular pattern image.
% OR
% >> filtr(MATRIX)
%    where MATRIX is the double-type matrix of
%    the cellular pattern image, i.e.
MATRIX=double(imread('cell.jpg'));
%
% You can also specify the HEIGHT, the WIDTH, and the center
coordinates Xc and Yc
% of your image. If you do so, type:
% >> filtr(...,HEIGHT,WIDTH,Xc,Yc)
% Enter either 1 argument (the image or its matrix), or 5.
%
% >> FILTERED=filtr(...) returns as FILTERED the filtered cellular
pattern matrix
% (CAUTION: it's not the PSD matrix!).


if isstr(MATRIX)
   MATRIX=double(imread(MATRIX)); % Turns the image into a matrix
end
% The previous loop allows us to input the image either as a function
argument
% (then CELLIMAGE is a string, for instance cell.jpg),
% or the image matrix (then CELLIMAGE is a matrix).

if nargin==1
   FINAL=calcpsd(MATRIX);
   [HEIGHT WIDTH]=size(FINAL);
   [Xc Yc]=center(FINAL);
end

if nargin~=1 & nargin~=5
   error (' Number of expected input arguments: 1 or 5');
end


% The difficulty is to paint black at the right location of the PSD,
% but we can't do it on the actual PSD, since it is then impossible
% to come back to the image matrix (because of .^2, log,... operators).
% So we process the filter on the NONFINAL matrix, whose pixels are
% on the same location as in the PSD (with of course not the same
intensity),
% but which allows us to come back to the image afterwards.

m=0;

WIND=MATRIX.*fspecial('gaussian',[HEIGHT WIDTH],0.2*min(HEIGHT,WIDTH));
% Windows the image with a gaussian filter
WIND=grayscale(WIND); % Rescales the entries between 0 and 255.
NONFINAL=fftshift(fft2(WIND));

while m~=2

   disp(' ');
   disp(' Please enter the angles of the 2 legs of the PSD :');
   TH1=2*pi/360*input('\n Positive angle (in degrees) = ');
   TH2=2*pi/360*input('\n Negative angle (in degrees) = ');

   g=input('\n Surviving gap = ');

   g1=g/2/cos(TH1);
   g2=g/2/cos(TH2);

   sm=menu('What area would you like to erase?','Around 3 and 9
o''clock', 'Around 6 and 12 o''clock','Around 3,6,9 and 12 o''clock');

   MASK=ones(HEIGHT,WIDTH);

   I=0;
   h=waitbar(0,'Please wait, filtering in progress...');

   for Y=1:WIDTH
      waitbar(I/WIDTH,h)
      I=I+1;
      X1=Xc-(Y-Yc)*tan(TH1);
      X2=Xc-(Y-Yc)*tan(TH2);

      for X=1:HEIGHT

         if (sm==1 | sm==3) & ((Y<=Yc & X2+g2<X & X<X1-g1) | (Y>=Yc &
X1+g1<X & X<X2-g2)), MASK(X,Y)=0; end
         if (sm==2 | sm==3) & ((X<X2-g2 & X<X1-g1) | (X1+g1<X &
X2+g2<X)), MASK(X,Y)=0; end

      end
   end

   close(h)

   MASKED=MASK.*NONFINAL;
   FILTERED=real(ifft2(ifftshift(MASKED)));
   LOCALFINAL=grayscale(calcpsd(FILTERED));


figure;colormap(gray);imshow(LOCALFINAL);image(LOCALFINAL);title('Filtr
-- Filtered Power Spectral Density');axis on;zoom on
```

```
   figure;colormap(gray);imshow(FILTERED);image(FILTERED);title('Filtr
-- Filtered Cellular Pattern');axis on;zoom on % Imshow>>Image gives
better results than Image alone. I don't know why...

   m=menu('What would you like to do now?','Try again','Quit or return
to main menu');

end

return
```

matlabsobel.m

```
function EDGED=matlabsobel(MATRIX)

% Edge enhancement using the Sobel method.
%
% EXAMPLES:
%
% >> matlabsobel('cell.jpg')
%    where cell.jpg is the cellular pattern image
% OR
% >> matlabsobel(MATRIX)
%    where MATRIX is the double-type matrix of
%    the cellular pattern image, i.e.
MATRIX=double(imread('cell.jpg')).
%
% >> EDGED=matlabsobel(...) returns as EDGED the matrix of the
% enhanced edge image.


if isstr(MATRIX)
   MATRIX=double(imread(MATRIX)); % Turns the image into a matrix
end
% The previous loop allows us to input the image either as a function
argument
% (then CELLIMAGE is a string, for instance cell.jpg),
% or the image matrix (then CELLIMAGE is a matrix).

[EDGED,t]=edge(MATRIX,'sobel'); % First try: Matlab selects its own
threshold.
EDGED=grayscale(double(EDGED));
figure;colormap(gray);imshow(EDGED);title(sprintf('Matlabsobel --
Enhanced edges with a threshold of: %d',t));axis on;

m=menu('What would you like to do now?','Change threshold','Quit or
return to main menu');

while m~=2
   sprintf(' Previous used threshold: %d',t)
   t=input(' Enter new threshold: ');
   EDGED=grayscale(double(edge(MATRIX,'sobel',t)));
   figure;colormap(gray);imshow(EDGED);title(sprintf('Matlabsobel --
Enhanced edges with a threshold of: %d',t));axis on;

   m=menu('What would you like to do now?','Change threshold','Quit or
return to main menu');

end

return
```

persobel.m

```
function EDGED=persobel(MATRIX)

% Personnal version of edge enhancement,
% using the Sobel method.
%
% EXAMPLES:
%
% >> persobel('cell.jpg')
%    where cell.jpg is the cellular pattern image
% OR
% >> persobel(MATRIX)
%    where MATRIX is the double-type matrix of
%    the cellular pattern image, i.e.
MATRIX=double(imread('cell.jpg')).
%
% >> EDGED=persobel(...) returns as EDGED the matrix of the
% enhanced edge image.


if isstr(MATRIX)
   MATRIX=double(imread(MATRIX)); % Turns the image into a matrix
end

%Mv=[-1 0 1 ; -2 0 2 ; -1 0 1];
%Mh=[1 2 1 ; 0 0 0 ; -1 -2 -1];
[HEIGHT WIDTH]=size(MATRIX);
GRAD=zeros(HEIGHT,WIDTH);
EDGED=zeros(HEIGHT,WIDTH);

%figure;imshow(MATRIX);colormap(gray);title('Cellular Pattern');axis
on;
%zoom on

h=waitbar(0,'Computing the image gradient, patience required!');

for I=2:HEIGHT-1 % This loop computes the gradient matrix GRAD.
   waitbar(I/HEIGHT,h)
   for J=2:WIDTH-1
      GRAD(I,J)=(MATRIX(I-1,J+1)+2*MATRIX(I,J+1)+MATRIX(I+1,J+1)-
MATRIX(I-1,J-1)-2*MATRIX(I,J-1)-MATRIX(I+1,J-1))^2+(MATRIX(I-1,J-
1)+2*MATRIX(I-1,J)+MATRIX(I-1,J+1)-MATRIX(I+1,J-1)-2*MATRIX(I+1,J)-
MATRIX(I+1,J+1))^2;
   end
end

close(h)

M=max(max(GRAD));
m=1;

while m~=2 % This loop compares the gradient to the input threshold.

   t=input('\n Enter threshold percentage (for instance 20): ');
   h=waitbar(0,'Comparing the gradient to the threshold, so far so
good...');

   for I=2:HEIGHT-1
      waitbar(I/HEIGHT,h)
      for J=2:WIDTH-1
         if GRAD(I,J)>t*M/100
            EDGED(I,J)=255;
         end
      end
   end

   close(h)

   figure;colormap(gray);image(EDGED);
   title(sprintf('Persobel -- Edge enhancement -- Threshold : %d
%%',t));
   axis on;zoom on;

   m=menu('What would you like to do now?','Change threshold','Quit or
return to main menu');

end
```

angplot.m

```
function angplot(FINAL,HEIGHT,WIDTH,Xc,Yc)

% Puts intensity as a function of angle at defined radius.
% The result is a plot of the intensity along a circle
% centered on the crossing of the PSD-legs, at the radius
% the user is invited to input.
%
% The computation processes the average intensity on a window
% whose size can also be specified by the user.
%
% You may want to display the PSD before, to help you visualize
% and choose the appropriate radius.
%
% The main purpose of this program is to know the position of the PSD-
legs.
% Once you know it, you can filter the PSD or run a radial scanning
(radplot)
% along one of the legs.
%
% EXAMPLES:
%
% >> angplot(FINAL)
%    where FINAL is the PSD matrix.
%
% You can also specify the HEIGHT, the WIDTH, and the center
coordinates Xc and Yc
% of your image. If you do so, type:
% >> angplot(FINAL,HEIGHT,WIDTH,Xc,Yc)
% Enter either 1 argument (i.e. FINAL), or 5.


if nargin==1
   [HEIGHT WIDTH]=size(FINAL);
   [Xc Yc]=center(FINAL);
end

if HEIGHT~=WIDTH
   disp(' WARNING: Your image isn''t square.')
   disp(' The result this program is going to return are WRONG...')
end

if nargin~=1 & nargin~=5
   error (' Number of expected input arguments: 1 or 5');
end


pi=3.14159265;

T=zeros(1,1024); % Intensity storage vector. Entries run through a
circle at radius R0.
BUF=[]; % Temporary storage array

m=0;
```

```
while m~=3

   if m==0 | m==1
      iwid=input('\n Enter pixel window half-width for smoothing : ');
      sprintf(' The averaging window size is now
%dx%d',2*iwid+1,2*iwid+1)
      sizeofwindow=(2*iwid+1)^2;
   end

   if m==0 | m==2
      R0=input(' Enter the radius (in pixels) of the circle \n you want
to run through : ');
   end

   h=waitbar(0,'Angular scanning in progress...not that easy!');

   for I=1:1024
      waitbar(I/1024,h)
      TH=pi/512*(I-512); % Angle of study, from -pi to +pi
      CT=cos(TH);
      ST=sin(TH);
      X=Xc-R0*ST; % Real coordinates of study
      Y=Yc+R0*CT;
      M=fix(X); % Top left closest pixel from the real point of study
      N=fix(Y);

      % CAUTION: the origin is at (1,1), in the top left corner of the
image.
      % That's why the expressions for X and Y are not those
expected...
      % Thus, the image is described in the trigonometric way,
      % beginning on the right-hand side of the center.

      if 1<=M & M<=HEIGHT & 1<=N & N<=WIDTH % Stay away from edges
         for II=M:M+1 % Calculation around 4 pixels surrounding the
real point
            for J=N:N+1
               LOCALSUM=0;
               for K=-iwid:iwid % Average of the window intensity
                  for L=-iwid:iwid
                     if 1<=II+K & II+K<=HEIGHT & 1<=J+L & J+L<=WIDTH %
Prevents the window from overlapping out of the image
                        LOCALSUM=LOCALSUM+FINAL(II+K,J+L)/sizeofwindow;
                     end
                  end
               end
               BUF(II,J)=LOCALSUM;
            end
         end

         % Now we'got 4 values : one on each pixel surrounding
         % the real point of study.
         % To synthetize those 4 values into 1, we will average them by
         % weighing them with the area of the opposite little square
         % around the real point (the closer the pixel is from the
point,
         % the more important it is in the average).
         P=X-M; % Lengths of the 4 squares
         Q=Y-N;
         PP=1-P;
         QQ=1-Q;

T(I)=PP*QQ*BUF(M,N)+P*QQ*BUF(M+1,N)+Q*PP*BUF(M,N+1)+P*Q*BUF(M+1,N+1);
      end
   end

   close(h)

   figure;
   I=1:1024;
   ang=180/512*(I-512);
   plot(ang,T)
   axis on
   xlabel('Angle in degrees, 0 is on the right hand-side of the
center')
   ylabel('Average intensity')
   title(sprintf('Angplot -- Intensity along the circle of radius %d
pixels -- Averaging window size : %dx%d',R0,2*iwid+1,2*iwid+1))
   zoom on

   m=menu('WHAT WOULD YOU LIKE TO DO NOW?','Change window averaging
size','Change the scanning radius','Quit or return to main menu');

end

return
```

radplot.m

```
function ANGL=radplot(FINAL,HEIGHT,WIDTH,Xc,Yc)

% Puts intensity as a function of radius at defined angle.
% The result is a plot of the intensity along a leg of the PSD,
% centered on the crossing of the PSD-legs, at the angle
% the user is invited to input (this angle is set as an output,
% in case it is needed afterwards).
%
% The computation processes the average intensity on a window
% whose size can also be specified by the user.
%
% You may want to run the program angplot before, to know the accurate
% position of the PSD-legs.
%
% EXAMPLES:
%
% >> radplot(FINAL)
%    where FINAL is the PSD matrix.
%
% You can also specify the HEIGHT, the WIDTH, and the center
coordinates Xc and Yc
% of your image. If you do so, type:
% >> angplot(FINAL,HEIGHT,WIDTH,Xc,Yc)
% Enter either 1 argument (i.e. FINAL), or 5.
%
% To get the angle of the scanned leg and store it in THETA, type:
% >> THETA=radplot(...);

if nargin==1
   [HEIGHT WIDTH]=size(FINAL);
   [Xc Yc]=center(FINAL);
end

if nargin~=1 & nargin~=5
   error (' Number of expected input arguments: 1 or 5');
end


pi=3.14159265;

T=zeros(1,2049); % Intensity storage vector. Entries run along a line
at angle TH0.
BUF=[]; % Temporary storage array

m=0;

while m~=3

   if m==0 | m==1
      iwid=input('\n Enter pixel window half-width for smoothing : ');
      sprintf(' The window averaging size is now
%dx%d',2*iwid+1,2*iwid+1)
      sizeofwindow=(2*iwid+1)^2;
   end
   if m==0 | m==2
      TH0=input(' Enter the angle (in degrees) of the leg \n you want
to run along : ');
      ANGL=TH0; % Just for the convenience. ANGL for the output, TH0 in
the program.
   end

   h=waitbar(0,'Radial scanning in progress...quite tough!');

   for I=-1024:1024
      waitbar((I+1024)/2048,h)
      R=sqrt((HEIGHT/2)^2+(WIDTH/2)^2)/1024*I; % Radius of study
      CT=cos(TH0*2*pi/360);
      ST=sin(TH0*2*pi/360);
      X=Xc-R*ST; % Real coordinates of study
      Y=Yc+R*CT;
      M=fix(X); % Top left closest pixel from the real point of study
      N=fix(Y);

      if 1<=M & M<=HEIGHT & 1<=N & N<=WIDTH % Stay away from edges
         for II=M:M+1 % Calculation around 4 pixels surrounding the
real point
            for J=N:N+1
               LOCALSUM=0;
               for K=-iwid:iwid % Average of the window intensity
                  for L=-iwid:iwid
                     if 1<=II+K & II+K<=HEIGHT & 1<=J+L & J+L<=WIDTH %
Prevents the window from overlapping out of the image
                        LOCALSUM=LOCALSUM+FINAL(II+K,J+L)/sizeofwindow;
                     end
                  end
               end
               BUF(II,J)=LOCALSUM;
            end
         end

         % Now we'got 4 values : one on each pixel surrounding
         % the real point of study.
         % To synthetize those 4 values into 1, we will average them by
         % weighing them with the area of the opposite little square
         % around the real point (the closer the pixel is from the
point,
         % the more important it is in the average).

         P=X-M; % Lengths of the 4 squares
         Q=Y-N;
         PP=1-P;
         QQ=1-Q;

T(I+1024)=PP*QQ*BUF(M,N)+P*QQ*BUF(M+1,N)+Q*PP*BUF(M,N+1)+P*Q*BUF(M+1,N+
1);
      end
   end

   close(h)
   figure;
   I=-1024:1024; rad=(sqrt((HEIGHT/2)^2+(WIDTH/2)^2)/1024)*I;
   plot(rad,T)
   axis on
   xlabel('Radius in pixels from the center')
   ylabel('Average intensity')
   title(sprintf('Radplot -- Intensity along the line of angle %d
degrees -- Averaging window size : %dx%d',TH0,2*iwid+1,2*iwid+1))
   zoom on

   m=menu('WHAT WOULD YOU LIKE TO DO NOW?','Change window averaging
size','Change the scanning angle','Quit or return to main menu');

end

return
```

peak.m

```
function RESULT=peak(DIMENSION,ANGL)

% Returns the sequence of the harmonics for the horizontal cell width.
%
% EXAMPLE:
%
% >> peak(DIMENSION,ANGL)
%    where DIMENSION is the dimension of the square image
%    and ANGL the angle of the PSD leg you've scanned.

m=2;

while m==2

   BUF=input('\n Enter the sequence of the coordinates of the peaks,
but not the DC! \n General syntax: [Abscissa#1 Ordinate#1;Abscissa#2
Ordinate#2; ... ] \n Example: [5 97;11 45;15 32] \n Now your turn: ');
   X=BUF(:,1)';
   Y=BUF(:,2)';
   SIZEPIXELPERP=DIMENSION./X; % Cell size perpendiculary to the cells.
   WEIGHT=(Y-max(Y))*(100-10)/(max(Y)-min(Y))+100; % Rescales the
energy of the sequence: energy of lowest peak = 10, energy of highest
peak = 100.
   SIZEPIXELHORI=SIZEPIXELPERP/cos(ANGL*2*pi/360); % Projects the
perpendicular cell width to the horizontal direction.

   disp(' ');
   disp(' Sequence of the harmonics for the horizontal cell size in
pixels:');
   disp(SIZEPIXELHORI);

   m=menu('Does it make sense?','Yes, convert into centimeters','No,
change values');

   if m==1
      disp(' ');
      disp(' Relation between pixels and cm -- Use Photoshop ruler!');
      c=input('    Enter a distance in cm: ');
      p=input('    Enter the corresponding number of pixels: ');

      SIZECMHORI=c*SIZEPIXELHORI/p;
      RESULT=[SIZECMHORI' WEIGHT'];
      disp(' ');

disp('**************************************************************');

disp('**************************************************************');
      disp(' ');
      disp(' Here''s the result:')
      disp(' 1st column: horizontal cell size in cm, and its
harmonics.');
      disp(' 2nd column: relative energy of each harmonic (linear
fit):');
```

```
      disp('    Energy of lowest peak = 10, energy of highest peak =
100');
      disp(' ');
      disp(RESULT);
      disp(' ');

disp('**************************************************************');

disp('**************************************************************');
      disp(' ');


   end

end

return
```

9- References

Shepherd, J. E., S. R. Tieszen (1986). “Detonation Cellular Structure and Image Processing”. *Sandia Report*, Sandia National Laboratories, Albuquerque, NM, SAND86-0033.

Shepherd, J. E., I. O. Moen, S. B. Murray, and P. A. Thibault (1986). “Analyses of the Cellular Structure of Detonations”. *21st Symposium (International) on Combustion*, The Combustion Institute, Pittsburgh, PA, 1649-1658.

Lee, J. J. , D. Garinis, D. L. Frost, J. H. S. Lee, and R. Knystautas (1995). “Two-dimensional autocorrelation function analysis of smoked foil patterns”. *Shock Waves (1995)* 5:169-174.