Automated analysis of microscopic images of cellular tissues

R.J. Slooter

Automated analysis of microscopic images of cellular tissues

by

Ir. R.J. Slooter

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday June 4, 2018 at 12:00 PM.

Student number:	4220943	
Project duration:	August 28, 2017 – June 4, 2018	
Thesis committee:	Prof. dr. ir. C. Vuik,	TU Delft, chair
	Dr. N.V. Budko,	TU Delft, supervisor
	Dr. J.W. van der Woude,	TU Delft
	s.s.t.t. R. Klooster,	HZPC, guest member

This thesis is confidential and cannot be made public until June 4, 2018.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Front: "Flooded Meadow", Tristram Paul Hillier (1949).
Glynn Vivian Art Gallery, Swansea (Abertawe), Wales, UK.

TU Delft

Preface

And when much people were gathered together, and were come to Him out of every city, He spake by a parable:
A sower went out to sow his seed: and as he sowed, some fell by the way side; and was trodden down, and the fowls of the air devoured it.
And some fell upon a rock; and as soon as it was sprung up, it withered away, because it lacked moisture.
And some fell among thorns; and the thorns sprang up with it, and choked it.
And other fell on good ground, and sprang up, and bare fruit an hundredfold. And when He had said these things, He cried, he that hath ears to hear, let him hear.

Luke 8:4-8

In the process of analysing potato crops a visual inspection is performed. In this analysis images of cellular tissue are examined to determine certain cell parameters. As this is a time consuming process the HZPC company is seeking an algorithm that can be utilized to automate this work. Difficulties in establishing such an algorithm are for example, the presence of broken cells and other image contaminating factors. This eventually leads to the failure of standard image processing tools, like boundary detection, to produce good results. The two-dimensional Fourier transformation provides us with an indicator for average cell size, but is lacking in more detailed information. So this leaves us with a task to establish a method that can provide a greater range of detailed cell parameters.

In this research project we investigate a variety of different approaches. A key element in our endeavour is the localising of points of interest (POI's). These POI's are the approximate centre points of the cells, the corners or vertices of the cells, and also the cell walls in given directions. Using these POI's we have built some intuitive methods, but we have also incorporated them in a contour finding technique called 'snakes'. The method concerning the snakes proved to be very fruitful, therefore we have studied a collection of greedy methods but also a gradient vector field method. Using the detected cells we can calculate the cell size and shape, which can be included in statistical analysis. From this statistical analysis we hope to provide a useful tool to quickly extract the data that is necessary to perform more thorough phenotype-genotype research.

Here I also want to give special thanks to HZPC for providing us with financial support, and the microscope images, and this interesting challenge.

R.J. Slooter
Delft, May 2018

Contents

1 Introduction

In this report we will build up to an algorithm that locates cells in microscopic images of cellular tissues. Our initial investigation is founded on the thesis by Peter Iles [1] and the 2014 paper by Liu et al. [2]. We will start out using intuitive methods, from there we will go into more formal mathematical descriptions and methods.

First we sketch the problem, let us do so by looking at one of the many microscope images we have in Fig. 1.1. From this figure we quickly learn that there are many different cells, and it is quite evident that the impurities can cause problems when analysing the image automatically.

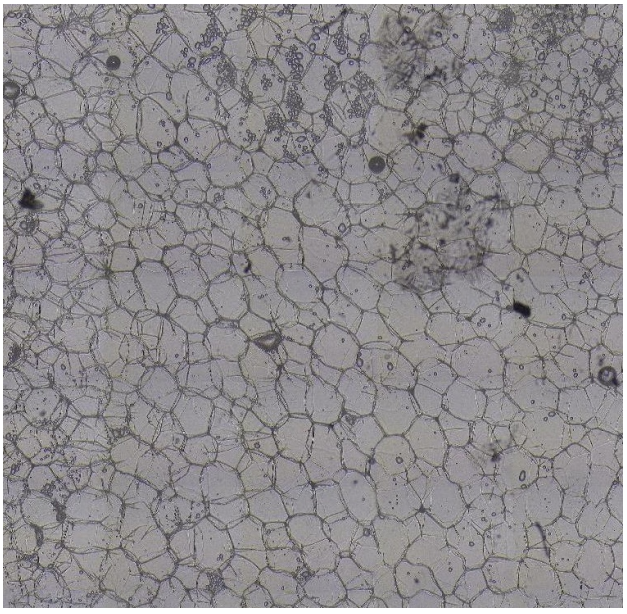


Figure 1.1: Cropped microscope image, in this image we see various cells and impurities. (This image, as well as all the other microscope images were kindly provided by HZPC.)

The challenge therefore is to make an algorithm that ignores the impurities as much as possible without destroying real data. Here we see the statistical hypothesis testing dilemma, we do not want false positives, but we do not want to throw out good cells either.

In order to get a better feeling for the image analysis and possible obstacles and issues we first look at the images using standard segmentation tools, e.g. skeletonisation and watershed. This means we do a superficial literature study on these topics, cf. [3, 4, and more]. Given the impurities it is very difficult to use these standard methods. We will describe these techniques, and show where they break down. This can be found in Chapter 2.

Then in Chapter 3 we will go into to study of the Weierstraß transformation. As it turns out finding the interesting points in the image can be done using a fairly simple filter, namely, the Gauß filter, which is commonly known as a blurring filter in the digital image processing world. We look at how we can use this particular filter to transform the image such that we can locate cell centres, but also cell vertices and cell boundaries. This information provides us with the anchor points for further algorithms.

After we have examined the POI's two basic intuitive ideas are worked out. In these we use the POI's to locate the cells. This is described in Chapters 4 and 5. First we describe a method in which we allow the computer to connect the vertices that were located. Here the underlying image brightness plays a role, we draw cell

walls over darker lines in the image. In the second method we use a model of fixed and free charges. The fixed charges are the located cell centres and vertices. The cells that we try to find are defined by free charges, which will act as cell vertices. These free charges are to flow into a stable position which should be the cell we want to locate. Then, in Chapter 5, we look at a formalised method to find contours, namely the active contour or snake algorithm [5]. This algorithm seeks to minimise the energy functional in Eq. (1.1).

$$E(\mathbf{r}) = \int_a^b \left[\alpha(s)E^{\mathrm{cnt}}(s) + \beta(s)E^{\mathrm{crv}}(s) + \gamma(s)E^{\mathrm{img}}(s)\right] \mathrm{d}s. \tag{1.1}$$

Where E^{cnt} and E^{crv} are internal energies of the snake, namely the continuity and curvature energy respectively. And E^{img} is the external energy which is sourced in the image that is to be analysed. The snake algorithm can be computed greedily [6, 8, 9], but also globally using a gradient vector flow [10, 11]. We find that the active contour algorithms provide good results and therefore we will use their results to generate statistical data on the cell parameters.

Using the active contour models we can calculate some cell parameters, these results will be briefly discussed in Chapter 6. In this chapter the cell perimeter, cell area and isoperimetric quotient are examined for several images. We also construct some probability density functions for the area and isoperimetric quotient, based on the assumption that the cell perimeter is normally distributed.

In Appendix A we will look at some basic theoretical background. For instance we revisit the convolution operation on a discrete grid. We will also look at the Gauß or Weierstraß transformation, which we will use to locate points of interest (POI's). We will also refresh our memory on the minimisation of a functional, which we need in Chapter 5.

The analysed images (bare and segmented) are provided in the Appendix (Appendix B), in Appendix C some code and practical remarks can be found.

2

Standard segmentation techniques

2.1. Introduction

In this chapter we will look into some of the commonly used techniques and study why they fail to segment the image properly. That is, in such a way that we can work with the resulting data. We are going to use some filters and segmentations, like for instance the watershed segmentation. We shall also have a brief look at some preprocessing methods, in particular the brightness enhancement tool.

Using these tools and techniques we can show what problems arise, but also look into advanced possibilities. Here we do not only look at our own work, but we will also consider some literature. From the cited material we hope to provide a better understanding of the segmentation methods, and show that there is a very wide range of possibilities.

2.2. Preprocessing

If we look at the given microscope images (e.g. see Figure 1.1), we can clearly distinguish some of the cells where others are not distinguishable at all. If we cannot decide where the cell is located how can we tell the computer what to look for? What we want to do here is enhance the image such that we have a general idea where the cell wall should be, because then we can guide the algorithm to return the right cells.

During our research we have used some preprocessing methods, we will look at the most important ones. First we look at the contrast enhancement using a sigmoid function, secondly we take a look at the thinning or skeletonisation algorithm and finally we will look at a corner detection algorithm. As before, for a concise overview we refer to *Digital Image Processing* [12]. We also use a cropping algorithm to identify and cut off bad edges of the images when necessary, and we use a resizing algorithm to reduce the size of extremely large images. These algorithms are described in Appendix C.

2.2.1. Contrast enhancement

Given the images the difference in brightness between cell walls and cell interiors is not very large. We can see the histograms of the pixel brightness in Figure 2.1.
In that figure we see that the difference in brightness of the cell walls and interior is rather small. So what we want is to increase the difference such that it is possible to make a much better distinction between cell walls and interiors. Hereto we use a sigmoid function, as is done in [1], which has the following shape.

$$I_{\text{out}}(x,y) = \frac{1}{1+e^{4m[I_{\text{avg}}(x,y)-I(x,y)+c]}}. \tag{2.1}$$

Here I is the original image, I_{avg} is a locally averaged version of the image, and I_{out} is the image with the enhanced contrast. The two scalars m, c are controlling the steepness and the offset respectively. Note its outcome is always between 0 and 1.

The behaviour of the sigmoid function is such that if the value of an element (x, y) of I is lower than $I_{\text{avg}}(x, y)+$

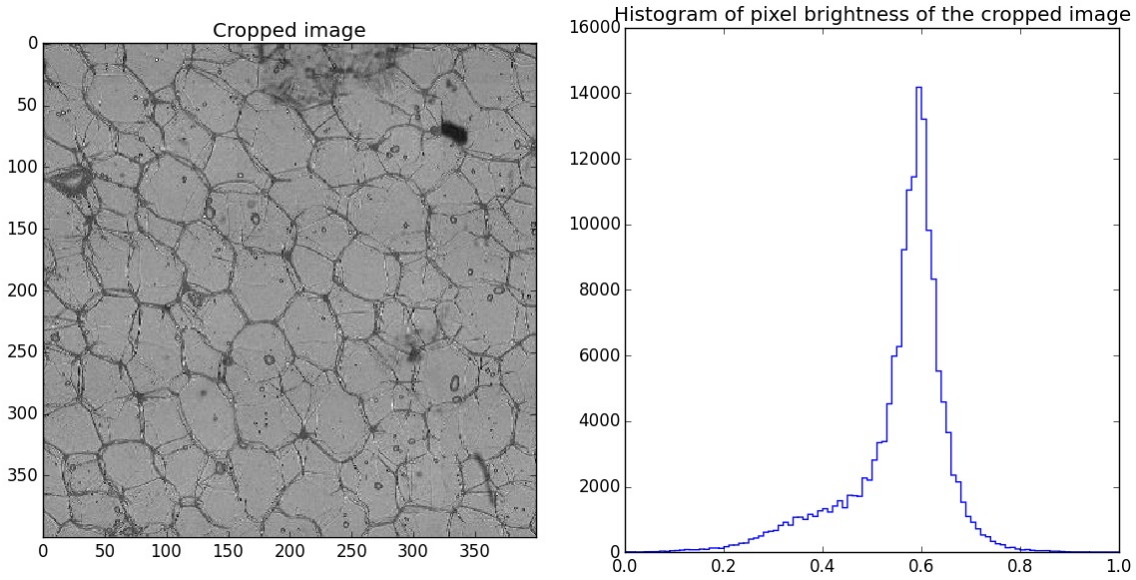


Figure 2.1: On the left we see an image and on the right a histogram of the pixel brightness in this image. It shows how the brightness of the light and dark features does not differ a lot, making it harder to distinguish between them.

c it becomes smaller than 0.5. And it becomes larger if the opposite is true. Now if m is very large a small difference between I and $I_{\text{avg}} + c$ creates a larger contrast difference.

The local average is calculated using a uniform circular mask, i.e. some array containing a centred circular shape filled with constant non-zero values and zeros around it. The non-zeros should have the value of one divided by the total number of non-zeros. An example can be found in Figure 2.2, in this figure we clearly see that the contrast is strongly enhanced. However we must point out that there are still a lot of impurities present. Some of these impurities are caused by the contrast enhancement.

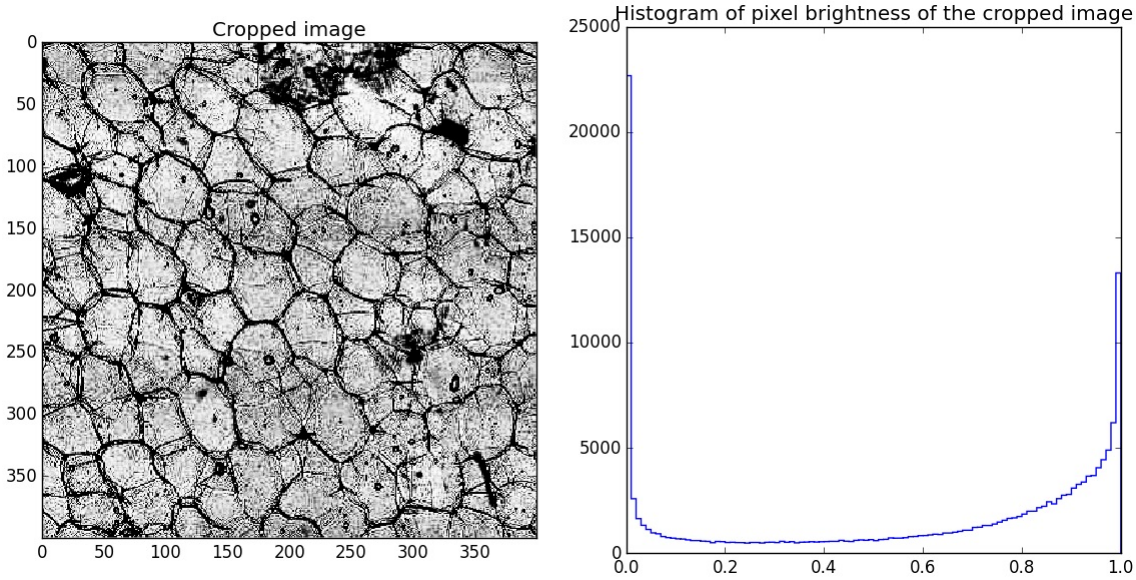


Figure 2.2: On the left we see the same area in in Figure 2.1, but now with enhanced contrast. We use a local average filter with a circular radius is 60 pixels, $m = 12$, and $c = 0.0$. Then on the right we get a histogram of the pixel brightness in this image. Clearly the difference between dark and bright pixels has increased dramatically. The negative side effect is that the impurities are also enhanced.

So with the contrast enhancement algorithm we can make the cell walls stand out more clearly, which means that it also improves the borders of different areas of the image. The latter being particularly useful when we use the watershed method. A further note is that the enhanced pixelation that comes out of the contrast

enhancement can be filtered using the Gaussian filter as described in the Appendix A, or some other low pass or local averaging filter.

2.2.2. Thinning algorithm

The second preprocessing method that we have examined is the thinning or skeletonisation algorithm. This algorithm operates on a binary image and thins the dark structures until only a single line remains, i.e. its skeleton. This is fully explained in [12, Chp. 9.4, pp. 194]. The essential takeaway is that all of the structures are "conditionally" eroded. Which means that all the structures are thinned until they have a maximum thickness of one pixel, while also keeping connected structures connected.

We have tried to use this filter in order to improve the use of the watershed algorithm as this removes the wide cell wall areas and makes sure they are not misidentified as cells. If we look at the same sample image as in Figure 2.1 and apply the contrast enhancement as described above. We round the pixel brightness to zeros and ones, then we use a skeletonisation function from scipy. The result is in Figure 2.3.

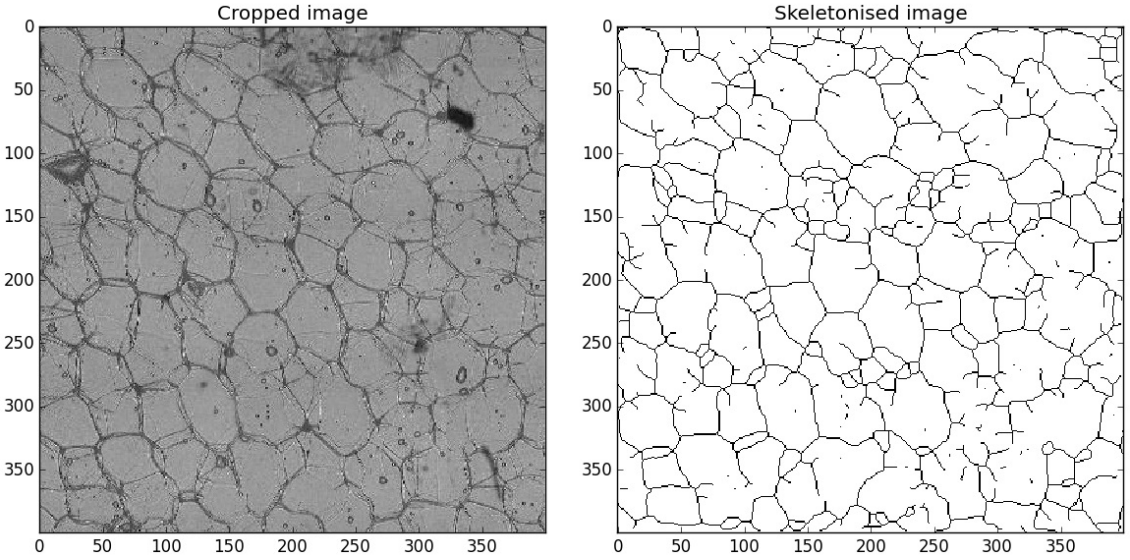


Figure 2.3: The left image shows the cropped image that we have used in earlier examples. On the right we see the result of enhancing and filtering the image, and ultimately skeletonising the image. Note that not all the cells from the left image are closed areas, as some walls are only faintly visible. Therefore they were removed in the processing steps. Still some cells are clearly visible, which is what we want for the watershed segmentation.

This preprocessing tool was particularly useful to make the image suitable for the watershed method. We can see that some small closed loops are formed, these will form false positives when we count each area as a cell. Secondly, some cells are merged together as there is no clear border between them. All-in-all, the improvements made on the one hand are weighed down by the newly generated impurities.

2.2.3. Corner finding algorithm

The final preprocessing tool we want to look at is the corner finding algorithm, that is in particular the Harris corner detection algorithm. Again we refer to [12, Chp. 7, pp. 148], in which the algorithm is worked out in detail. The general idea is that the corners in the image contrast strongly with their surroundings, i.e. the 'derivative' in the x and y direction is large. So when we find such a point in the image we assume it is a corner.

We want to find these corners for the shape fitting that we want to perform, which is described in Chapter 4. In the shape fitting we need valid data points, instead of all the pixel data. We therefore will assume that the corner finding algorithm will provide us with the data points that we can use.

In Figure 2.4 we see the result of the corner finding on the image with enhanced contrast. Some corners and cell walls have been found correctly, but we also note that some of the identified corners are actually impurities in the image. In the shape fitting these false corners corrupt the actual data that we want to fit our

cells with. This is harmful for the fitting of straight lines through a given set of points, or for fitting ellipses. In particular when we use a small data set, because each data point has equal influence on the result.

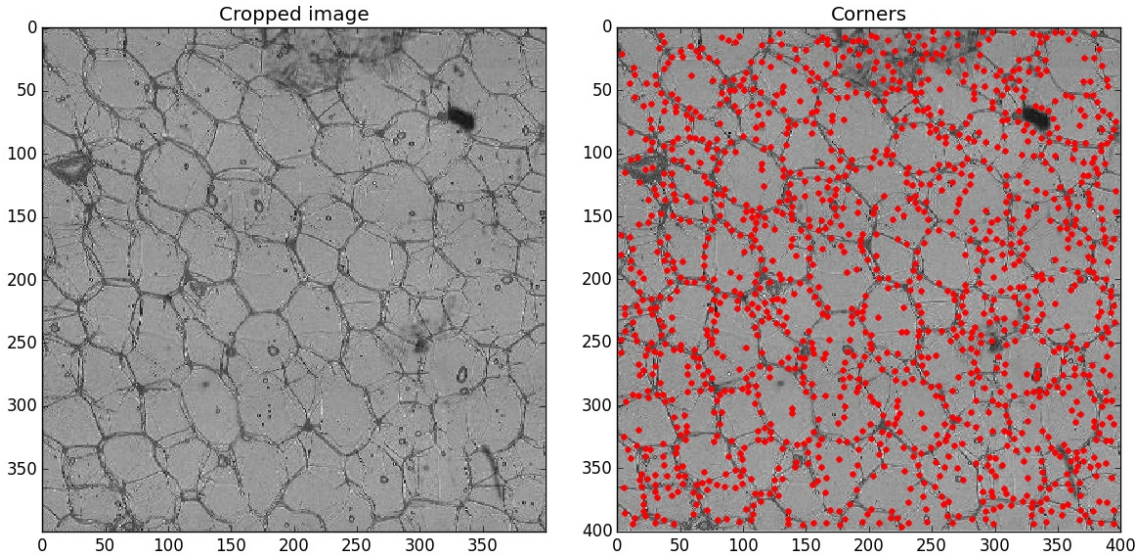


Figure 2.4: The left image shows the cropped image that we have used in earlier examples. On the right we see, in red dots, the corners we have found. We have preprocessed the image by enhancing the contrast of the left image.

2.3. Watershed segmentation

Now that we have explained some of the preprocessing steps we can attempt the segment the image using some basic technique. One such technique is the watershed method. In this method we locate some drainage points, preferably one per cell. The problem is that we do not know where these drainage points are. For simplicity we can choose a square grid of points. In this grid we presume that there is a drainage point every 20 pixels and let the computer eliminate drainage points that turn out to be in the same drainage area. Hereto we only use pre-made functions from libraries, we have not written a new code.

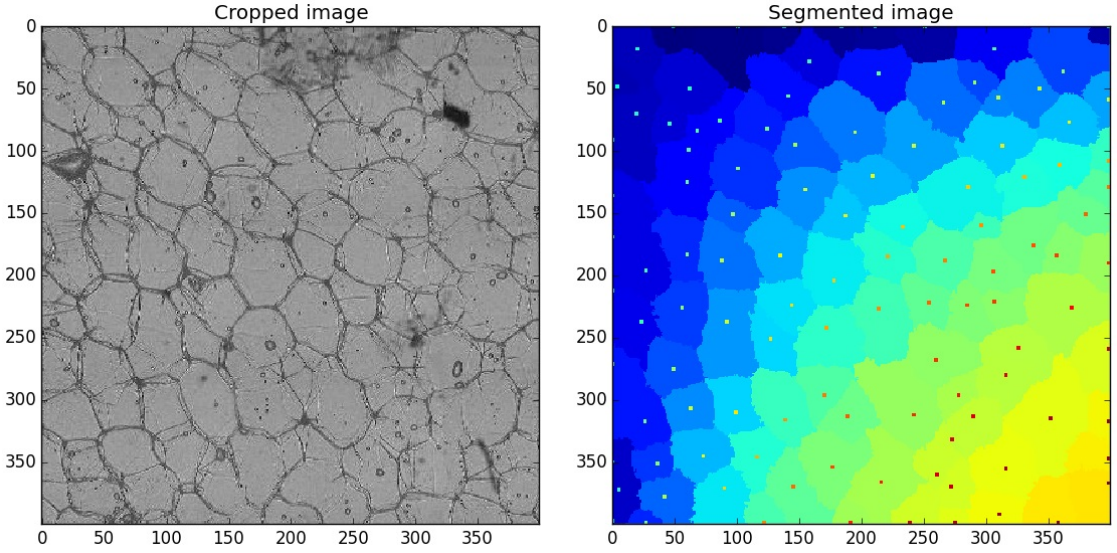


Figure 2.5: Again we see the cropped image on the left. In the right image panel we see the segmented image, in each segment the located cell centre is also drawn (these particular cell centres were found using the Gaussian filter as described in Chapter 3). Notice how irregular the edges of the segments are. On the upside the cells that are visible in the left image can be recognized quite well.

Given the result in Figure 2.5 we see that the cell walls are erratic, which is not what we expect. In short, the

results do not show us a good segmentation like we want. Notice for example how in the middle of the right side of the original image the cell walls are faint, the segmentation has found cells that do not resemble the original image. In addition to that we remark that the segmentation is not practical either as there is no structured list from which we can calculate geometric properties. Also note that here we have used cell centres that we have localised, rather than a square grid as suggested above. So even with a guided segmentation the results are not good.

At the end of this chapter we have learned that it is not easy to segment the image using a standard method, like the watershed as in [1]. We now know that we cannot simply work with the original images without performing any preprocessing steps.

3

Cell centre detection

3.1. Introduction

The most straightforward way to analyse an image is by looking at it as it is given. In this chapter we therefore intend to work with the given image, i.e. we will not go to the frequency domain. We have seen that the watershed method returns erratic cell boundaries, which is not what one would expect when looking at the image, therefore we want to create a fixed reality in which we can have confidence. One takeaway from the watershed method is that we need good drainage points, which is what we will look at in this chapter. We want to find our so called points of interest (POI's), e.g. cell centres, vertices and boundaries. We will also assume that in an idealised world the cells are of a hexagonal shape.

3.1.1. The Gaussian filter and the Weierstraß transform

An important filter that we will use is the Gaussian filter. This filter is a convolution with a Gaussian function, in image processing it is generally known as a blurring filter. However it can also be used to find cell centres, vertices and boundaries in our particular problem.

Definition 3.1.1. Mathematically the convolution with a Gaussian function is known as the **Weierstraß transformation**, which is given as:

$$W[f](x) \equiv F(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} f(y) e^{-\frac{(x-y)^2}{2\sigma^2}} dy = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x-y) e^{-\frac{y^2}{2\sigma^2}} \mathrm{d}y, \tag{3.1}$$

where σ is the standard deviation of the Gaussian.

Take note that we obviously use it in a discrete setting rather than the continuous, this as we are dealing with individual pixels.

Another important remark about the Weierstraß transform is that it acts as a low pass filter, this as signals with a high frequency get a bigger reduction than low frequency signals. In other words, a small feature in an image gets removed more quickly than a large structure. This is a very important property that has its effect on the actual image analysis as it allows us to filter our small impurities easily.

Using this and a binary maximum filter (see Section A.2.2) we are able to find local maxima in the image brightness. As a secondary result it can be used to find local minima on the inverse image. And if we use a different mask we can find local maxima in 1D rather than in 2D, thus allowing us to find minimal lines in the image. All of these results are very useful in the later image analysis as we shall see in Chapters 4 and 5.

3.2. Cell centre localisation

In order to find cells it is key that we can determine the central point of the cells in the image. As this central point can be used as a marker. In order to do that we have opted to use a Gaussian filter, hereto we need to study its behaviour in more detail to select a good filter that returns the actual centre of each cell.

We recall Equation (3.1), bear in mind that we do not really operate on a continuous space. This however does not change the argument. The Gaussian filter or Weierstraß transform blurs the image, this as the image is convolved with a Gaussian weighted mask. The standard deviation, or σ, of the Gaussian determines the degree of blurring, but is also a key component in tailoring a filter that locates the cell centres.

If we put it in practical terms, we try to determine the cell centres using the brightness of the grey scale image. As the interior of cells is much brighter than the cell walls we try to find local maxima in brightness. We must consider how the value of σ affects the resulting image. A small standard deviation will not blur the image a lot, hence the signal noise contains local maxima and will consequently yield many false positives for cell centres. On the other hand a large standard deviation will blur the image a great deal, which in turn will completely blur out small cells and will throw away good cell centres. Therefore it is important to make an intelligent choice rather than selecting some arbitrary number for σ.

To make a decent choice we must do a theoretical study into the behaviour of Gaussian functions, in particular we want to know when two Gaussian peaks can be resolved and when they no longer have two distinct maxima. One can compare this, for illustrative purposes, in essence to the resolving of two peaks in optical diffraction. We can no longer find the two peaks when they merge into one, which is somewhat similar to the Sparrow criterion (cf. *Optical physics* [19, Chp. 12.2.2.]). However we are not dealing with the sinc function, but with Gaussian functions.

3.2.1. Separating two equal Gaussian functions

We analyse the easiest trivial example, i.e. the Weierstraß transform of two Dirac delta functions. The two delta functions are separated by some distance w, the resulting transformation is nothing more than two Gaussian functions centred around the original positions of the delta functions. This can straightforwardly be derived from Eq. (3.1). The result of the transformation for different values of σ can be seen in Figure 3.1, we define a function $C(x)$ in Eq. (3.2) as follows:

$$C(x) = \frac{1}{\sigma\sqrt{2\pi}}\left(e^{-\frac{1}{2}\left(\frac{x}{\sigma}\right)^2} + e^{-\frac{1}{2}\left(\frac{x-w}{\sigma}\right)^2}\right). \tag{3.2}$$

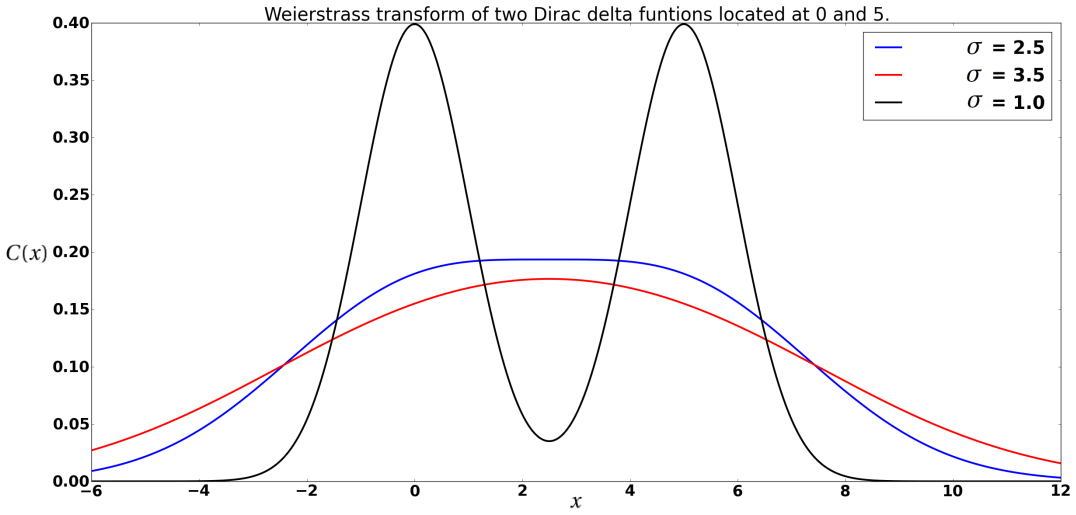


Figure 3.1: Weierstraß transform for three different standard deviations. The distance between the original delta functions is $w = 5$. We see that if σ is large there is only one maximum, and if σ is small there are two peaks. More importantly at one particular value the two peaks merge.

Now the remaining problem is to determine when the two peaks merge. Hereto we need to define some terms concerning the resolvability of the two maxima, cf. [20, 21]. From these papers we take the definition for the resolvability of resolution functions R_s and R.

$$R_s = \frac{\mu_2 - \mu_1}{2(\sigma_1 + \sigma_2)}, \quad R = \frac{\mu_2 - \mu_1}{4\sigma}. \tag{3.3}$$

Where R is just the resolution function for two Gaussian functions with the same σ, secondly μ_i is the centre of the Gaussian. Then for us, as we choose to separate the two Gaussian functions by some distance w, and also the σ must be the same our resolution becomes:

$$R = \frac{w}{4\sigma}. \tag{3.4}$$

NB. in this case the two Gaussian functions have an equal weight.

The two peaks are no longer separable when $R \leq 0.5$, which is the case when $\sigma \geq \frac{w}{2}$. This can also be established from the fact that the derivative of the sum of the Gaussian functions has exactly one zero if $\sigma \geq \frac{w}{2}$. Let us take the derivative of Equation (3.2):

$$\frac{dC(x)}{dx} = -\frac{1}{\sigma\sqrt{2\pi}}\left(\frac{x}{\sigma^2}e^{-\frac{1}{2}\left(\frac{x}{\sigma}\right)^2} + \frac{x-w}{\sigma^2}e^{-\frac{1}{2}\left(\frac{x-w}{\sigma}\right)^2}\right). \tag{3.5}$$

Setting this equal to zero is the same as solving:

$$xe^{-\frac{1}{2}\left(\frac{x}{\sigma}\right)^2} + (x-w)e^{-\frac{1}{2}\left(\frac{x-w}{\sigma}\right)^2} = 0. \tag{3.6}$$

Which has a solution at $x = \frac{w}{2}$, regardless of the value of σ. Now to prove that there is only one solution if $\sigma \geq \frac{w}{2}$ we rewrite the equation. First we take the logarithm of the equation and reorder.

$$\frac{w^2 - 2xw}{2\sigma^2} = \ln\left(\frac{w-x}{x}\right). \tag{3.7}$$

The left hand side is just a straight line, and the right hand side is a curve that only exists for $0 < x < w$. Now these two curves always intersect at $x = \frac{w}{2}$. Now to see how many intersections there are we can take the derivatives of both lines and set them equal to find:

$$\frac{1}{\sigma^2} = \frac{1}{x(w-x)} \tag{3.8}$$

Notice that if $\sigma = \frac{w}{2}$ the slope and function value are equal in the point $x = \frac{w}{2}$. If $\sigma < \frac{w}{2}$ the slope of the left hand side is steeper at the point of intersection, but there are two points at which the slope is equal. Given that the right hand side slope goes to $-\infty$ when the edges of the domain are reached. The left hand slope is negative too, this means that the function will have two additional points of intersection from the *Mean value theorem* (see [22, pp. 136-142]). Now if $\sigma > \frac{w}{2}$ there are no points at which the slopes are equal so then we only have the one point of intersection at $x = \frac{w}{2}$.

So now we find that there exists exactly one merging point for two equally weighted Gaussian functions that are separated by some distance w at $\sigma = \frac{w}{2}$. A larger standard deviation only merges the two peaks more into one.

3.2.2. Separating two Gaussian functions of different magnitude

The results for the equal peaks are satisfying, however given the nature of the images that we will be assessing we also want to know when two unequal peaks are no longer resolvable. We define the sum of two Gaussian functions, now m is some value between 0 and 1. It describes the ratio between the peak heights. Then we obtain an expression similar to Eq. (3.2) we get:

$$C(x) = \frac{1}{\sigma\sqrt{2\pi}}\left(e^{-\frac{1}{2}\left(\frac{x}{\sigma}\right)^2} + me^{-\frac{1}{2}\left(\frac{x-w}{\sigma}\right)^2}\right). \tag{3.9}$$

The computation required to find the desired value for σ is now much more complicated, as the centre point *i.e.* the local minimum between the peaks, is no longer fixed. However we get an approximation from [20]:

$$R_{ms} = \frac{sR_{1s} + \sqrt{R_{1s}^2 + \frac{1}{2}\ln m}}{1+s}. \tag{3.10}$$

Here s is the ratio between the σ's of both peaks, which in our situation will always be one so we get:

$$R_m = \frac{R + \sqrt{R^2 + \frac{1}{2}\ln(m)}}{2}. \tag{3.11}$$

Recall R from Equation (3.3). As said before when $R \leq 0.5$ we can no longer distinguish the peaks, then we find an approximation for the minimal σ:

$$\sigma \approx \frac{w}{2-\ln(m)}. \tag{3.12}$$

In the case $m = 1$ we are back in the already solved situation and we find $\sigma = \frac{w}{2}$ again. However this is an approximation, the real value can be found by solving the equation:

$$e^{-\frac{1}{2}\frac{w\sqrt{w^2-4\sigma^2}}{\sigma^2}} = m\frac{w-\sqrt{w^2-4\sigma^2}}{w+\sqrt{w^2-4\sigma^2}}. \tag{3.13}$$

Which proves to be very difficult to solve in general, however we can find a numerical solution when we know the value for w. A contour plot of the solution is given in Figure 3.2.

Given all this information we can make some observations when analysing the image using the Weierstraß transform. In particular, as we will see in the next section, we no longer are bound to merely arbitrary choices that 'seem to do the right thing'. We can now make some inferences on the image content based on the results from the Weierstraß transform when we perform it for various value of σ. Moreover we will be able to make a more insightful choice for σ. Here it should be mentioned that still, due to the nature of the images, the choice is not necessarily correct for one particular value of σ. This since there are many different sized features in the image, so a certain trade-off needs to be made between removing small cells and oversampling large cells (i.e. picking multiple centres in one cell).

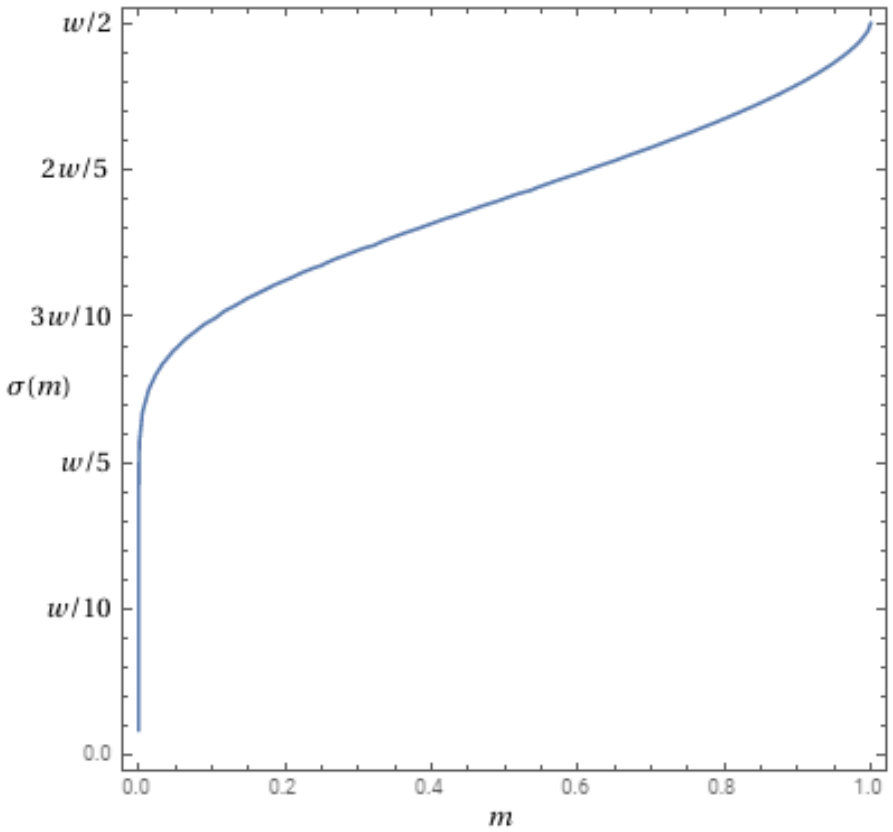


Figure 3.2: The contour plot showing the numerical solution for Eq (3.13).

3.2.3. Analysis of a hexagonal grid

Using the Weierstraß transform and the gained insights *ut supra*, we can now analyse the real world effects on an idealised figure. Performing an analytical dissection of a 2D image is, as can be imagined, difficult. We therefore will show the results by a *demonstratio ad oculos*, where we use a small example image containing a hexagonal pattern with some added features for illustrative purposes. We see the result in Figure 3.3.

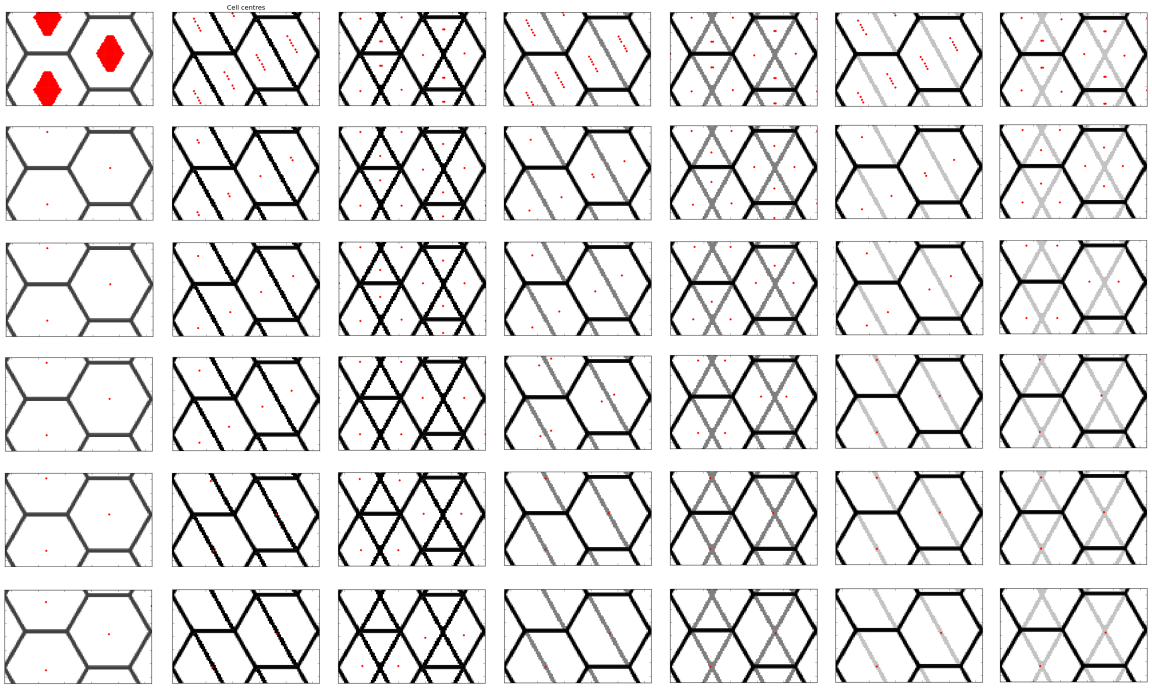


Figure 3.3: When we try to find the 'cell' centres on this hexagonal pattern we see how a different σ changes the outcome. In the first row $\sigma = 3$, then it gets progressively larger, $\sigma = 3, 5, 7, 10, 15, 25$. When sigma is too small too many centres are found, but when it is too large small cells get merged. Then in each column a different type of feature is added to the hexagonal pattern, varying in brightness and shape. Small cells will disappear when σ grows, and in particular weakly drawn cell walls vanish quickly.

3.3. Segmentation and vertices

Now using these centres we want to make the actual segmentation into different cells. The question that we now have to ask ourselves is, 'What do we want to do with the cell centres'. We could try to implement a watershed segmentation, or some other standard image segmentation method. After trying to do that we can only conclude that this is not the way forward for us, as it simply does not work like we want it to. Another obvious solution was staring us in the eyes all this time, we can also locate the vertices of the cell (given that we assume the cells to be polygons) by using a similar Weierstraß transform. Again we need to find a suitable value for σ, we shall base this value on the value that we use to find the cell centres.

3.3.1. Optimal filter parameter

We have established, in the previous section, that two peaks of equal intensity can be distinguished when $\sigma < \frac{w}{2}$. Using as basic an assumption as we can we will assume that the peaks we distinguish are indeed of equal intensity. Then we can easily calculate the value for σ that we need using basic geometry, we now explicitly use the assumption that the cells have a hexagonal shape.
Now from Figure 3.4 we deduce that the solution is just Pythagoras' formula:

$$\begin{aligned} \frac{w^2}{4} + \sigma^2 &= 4\sigma^2, \\ \frac{w^2}{4} &= 3\sigma^2, \\ \frac{w^2}{12} &= \sigma^2, \\ \Leftrightarrow \qquad \sigma &= \frac{w}{2\sqrt{3}}. \end{aligned} \tag{3.14}$$

Now we can find both the cell centres and the vertices, using just two Weierstraß transformations.

Actually we note that we could use any peak intensity, if we persist in our hexagonal assumption we can also uphold the Pythagorean approach, and just divide whichever σ we use for the cell centres by $\sqrt{3}$. From

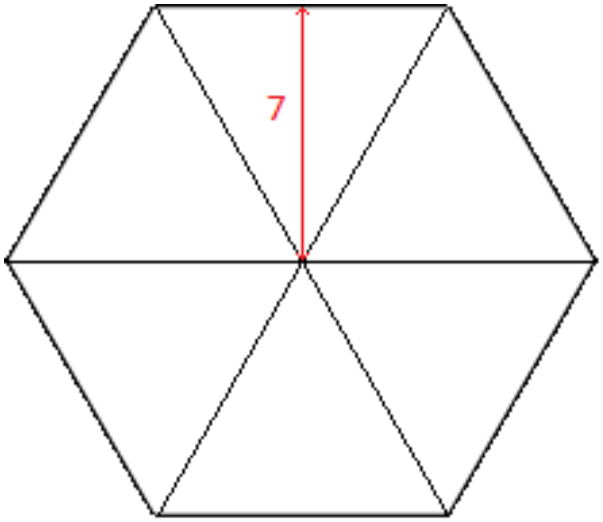


Figure 3.4: Using a hexagon we can add our knowledge of the cell centres. Using the standard geometry of the hexagon we are left with Pythagoras' formula.

here we can utilize several strategies involving these points of interest (POI's), we will look into this in Chapters 4 & 5.

Now the value of σ was determined by examining the number of located centres. We use the Gaussian filter with various values for σ, and count the number of centres. We plot the number of centres vs. the value of σ on a semilog scale (logarithmic for σ). We see that there is a domain of σ's for which the slope is small, we determine that the correct value is located in the middle of this domain.

4

Heuristic methods

4.1. Introduction

In this chapter we will draw on the points of interest (POI's) from Chapter 3. We have identified several paths to follow, to which we will look in detail. We start out with simple shape fitting methods, where the shapes are based on the centres that we have found. We continue with an intuitive method in which we just use the cell centres. For each centre we determine the cell as the area that is closer to it than to any other centre. Then our new approach involves using the original image, on the image we try to connect the located vertices by comparing their route with the darkness of the cell walls. This method looks very pleasing to the eye, but yields highly impractical results. In the final path we use a quasi-mechanical line of action. Here the POI's will represent fixed charges in the image space. This will give us a certain potential map on the image, on this map we add a certain number of free charges around a cell centre. The cell centres will be repelling the free particles, and the vertices will attract the free charges. As we let the system 'evolve in time', an equilibrium is found which is a cell.

4.2. Minimal distance

An intuitive method that we look into is a minimal distance method. Here we use the cell centres to build a new 'secondary' cell grid. Between all the centres we divide the image up in the areas that are closest to each centre.

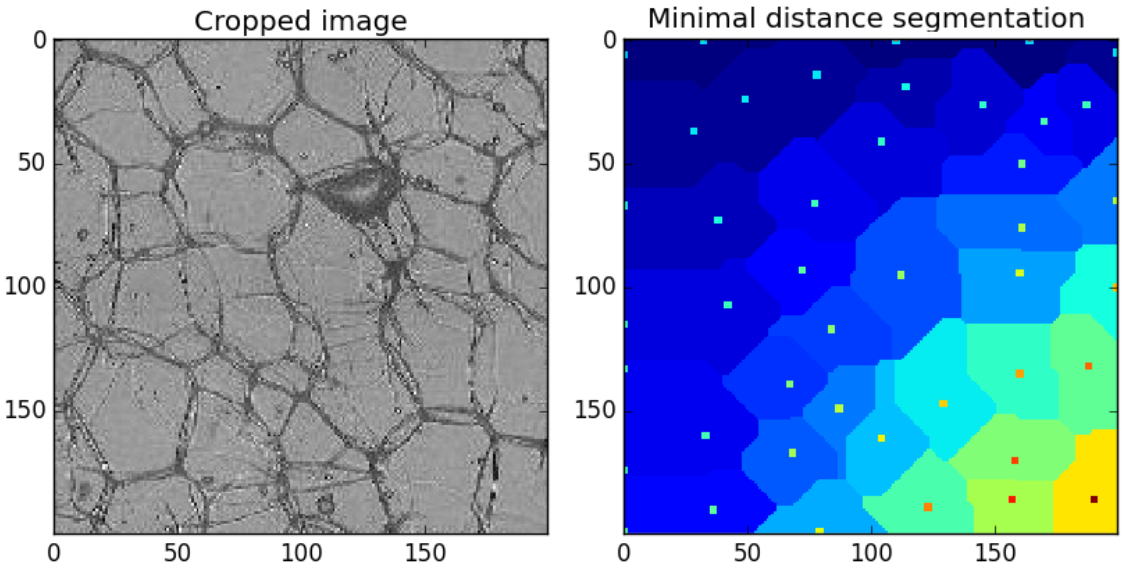


Figure 4.1: In the right image the segments are coloured according to the minimal distance the a cell centre. When we compare the image content to the cropped image on the left it is clear that a lot of information is lost.

This gives a very smooth segmentation as seen in Figure 4.1, but there are multiple problems. First of all, we have deleted all the initial knowledge of the image and are left with just the cell centres. This means that a small cell now may have grown tremendously at the expense of a large cell. In terms of the average size this is inconsequential but for the spread of the data it changes everything. So from that perspective we do not want to use this method at all. Furthermore some areas in the original image do not belong to any detected cell centre, like the dark structure in the middle. This is not removed from the newly found cells, which is bad as we do not want to over estimate the average cell size either.

So all in all, the resulting image looks very nice, but the price to pay is that the geometric data of the cells is gone. And the whole purpose of the project is to determine these geometric parameters of the cells as accurately as possible.

4.3. Shape fitting

4.3.1. Polygon

When we look at the cells, fitting a polygon seems a quite natural response. Most of the cells are polygons, and in particular they are hexagonal in shape. From the watershed method we have received rough edged cells, so we wanted to improve them by fitting a straight line through the boundary pixels. As we assume a hexagon we fitted an octagon, from the resulting octagon we can always remove two edges when necessary. This also allows use to use eight fixed directions instead of six random directions which would make the programming difficult.

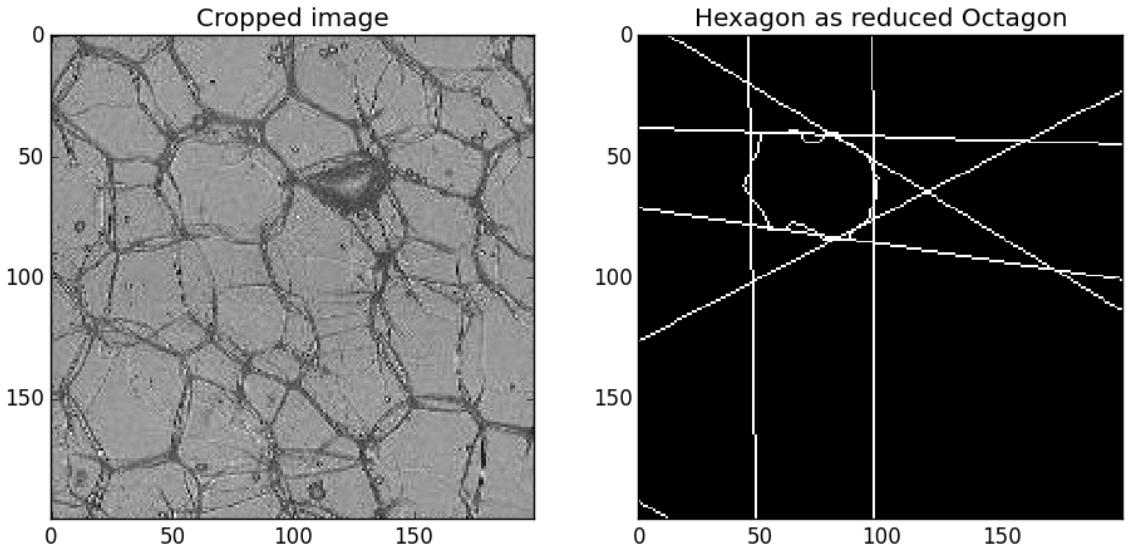


Figure 4.2: As before we see the cropped image on the left. In the right image panel we see one selected cell on which a polygon is fitted, this is a hexagon reduced from the initial octagon that was fitted to the data.

The result, see Figure 4.2, is not bad at all, but this requires a lot of work for just one cell. Also bear in mind that this cell already had a good shape to start with, when we look at poorly segmented cells this will give bad approximations.

So we want to do something else. Without performing the watershed, which costs time and does not give a better starting point, we have opted to use brightness scale of the pixels. From the pixel brightness we have made a weighted data set. The dark cells then have the greatest weight and the brighter cells are less weighty. Then dividing the cell that we want to fit in eight sectors defined by the cell centre, and the centres of its eight nearest neighbours. (Or to be more specific the points exactly between two neighbouring centres).
The result (Fig. 4.3) does not leave any room for debate, it is a bad approximation. This is mainly due to the shape of the sector, an inward pointing triangle. So we want to decrease the number of bad data points.

We retain the sectors, but now we only count the corner points that were found using the Harris corner detection algorithm. The results however are of the same nature. Nevertheless this has lead us to a new idea

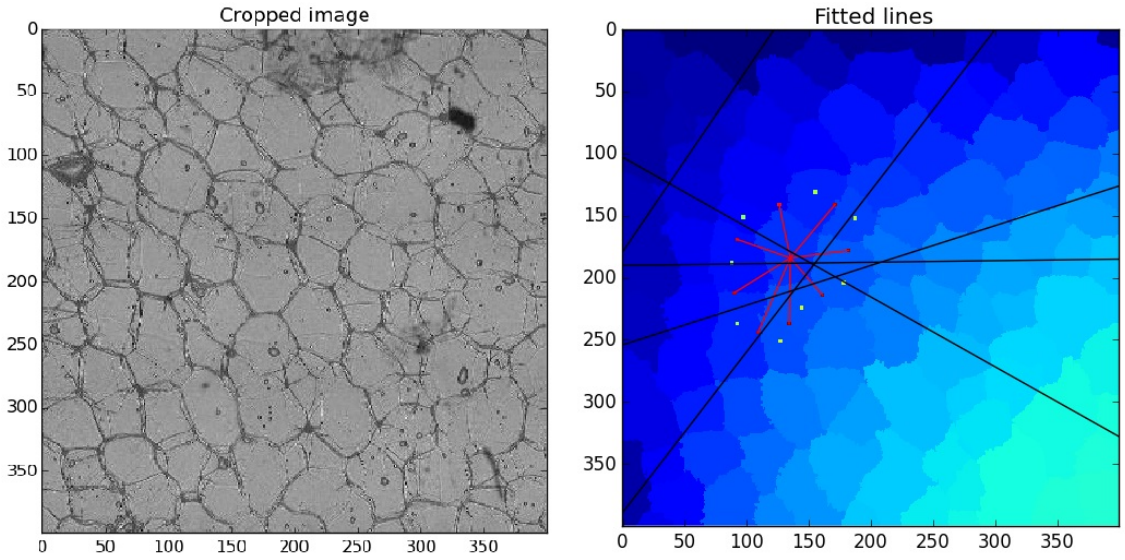


Figure 4.3: The cropped image is on the left. In the right image panel we see one selected cell on which a polygon is fitted, the fit is based on the pixel brightness. Clearly the result is not what we want at all.

that we have already worked out in Chapter 3. Namely we can determine cell vertices and use those to build our cells with. This as the use of the cell corners does provide a good basis to work with.

4.3.2. Ellipse

Another shape that approximates the cells very well is the ellipse. As a bonus it is easy to calculate the area, circumference, eccentricity and orientation of an ellipse. Here we could for instance directly use the corners that we found using the Harris corner detection algorithm, that is only those in our cell sectors.

The fitting method we use is worked out in [17, 18]. We notice that an ellipse can be defined as a set of points $\mathbf{x} = (x, y)$ with

$$f(\mathbf{a}, (x, y)) = \mathbf{D} \cdot \mathbf{a} = 0. \tag{4.1}$$

Where $\mathbf{D} = (x^2, xy, y^2, x, y, 1)$ and $\mathbf{a} = (a, b, c, d, e, f)$. Then an ellipse can be fit to a set of N data points: $\mathbf{x}_i, i = 1, \ldots, N$. This is done by minimising

$$\mathscr{D}(\mathbf{a}, \mathbf{x}) = \sum_{i=1}^{N} \left(f(\mathbf{a}, \mathbf{x}_i)\right)^2 = \mathbf{D} \cdot \mathbf{a} \Leftrightarrow \mathscr{D}(\mathbf{a}, \mathbf{x}) = \sum_{i=1}^{N} \mathbf{a}^T \mathbf{D}_i^T \mathbf{D}_i \mathbf{a} \equiv \mathbf{a}^T S \mathbf{a}. \tag{4.2}$$

Here $S = \sum \mathbf{D}_i^T \mathbf{D}_i$ is a 6×6 scatter matrix.

Then the goal is to find a vector $\mathbf{a}$ that minimises $\mathscr{D}(\mathbf{a}, \mathbf{x})$ such that the result yields an ellipse. Then we need that $4ac - b^2 > 0$ (or like the paper states $4ac - b^2 = 1$) in order for the solution to be an ellipse. In matrix notation this means $\mathbf{a}^T C \mathbf{a} > 0$, here C is a 6×6 matrix.

$$C = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{4.3}$$

Then as $f(\mathbf{a}, \mathbf{x}) = 0$ is invariant to linear scaling of $\mathbf{a}$. So we can rewrite the positivity constraint as $\mathbf{a}^T C \mathbf{a} = \varphi$, where $\varphi > 0$.

$$\text{argmin}_a \{\mathscr{D}(\mathbf{a}, \mathbf{x}) | \mathbf{a}^T C \mathbf{a} = \varphi\}. \tag{4.4}$$

Then the problem can be written in terms of Lagrangian multipliers or as a generalised eigenvalue problem.

For the Lagrangian multiplier method we introduce a multiplier λ and a Lagrangian

$$L(\mathbf{a}) = \mathscr{D}(\mathbf{a},\mathbf{x}) - \lambda(\mathbf{a}^T C\mathbf{a} - \varphi). \tag{4.5}$$

This Lagrangian needs to be minimised with respect to $\mathbf{a}$. Which is nothing more than taking the derivative with respect to $\mathbf{a}$ which gives the resulting equation.

$$S\mathbf{a} = \lambda C\mathbf{a} \Rightarrow \mathbf{a}^T S\mathbf{a} = \lambda \mathbf{a}^T C\mathbf{a} \Rightarrow \frac{\mathbf{a}^T S\mathbf{a}}{\mathbf{a}^T C\mathbf{a}} = \lambda. \tag{4.6}$$

So we need to minimize λ to minimise $\mathscr{D}(\mathbf{a},\mathbf{x})$.

We can also solve the generalised eigenvalue problem as we notice that $S\mathbf{a} = \lambda C\mathbf{a}$ can be rewritten as

$$\frac{1}{\lambda}\mathbf{a} = S^{-1}C\mathbf{a}. \tag{4.7}$$

When we solve this we want to find the largest eigenvalue $1/\lambda$.

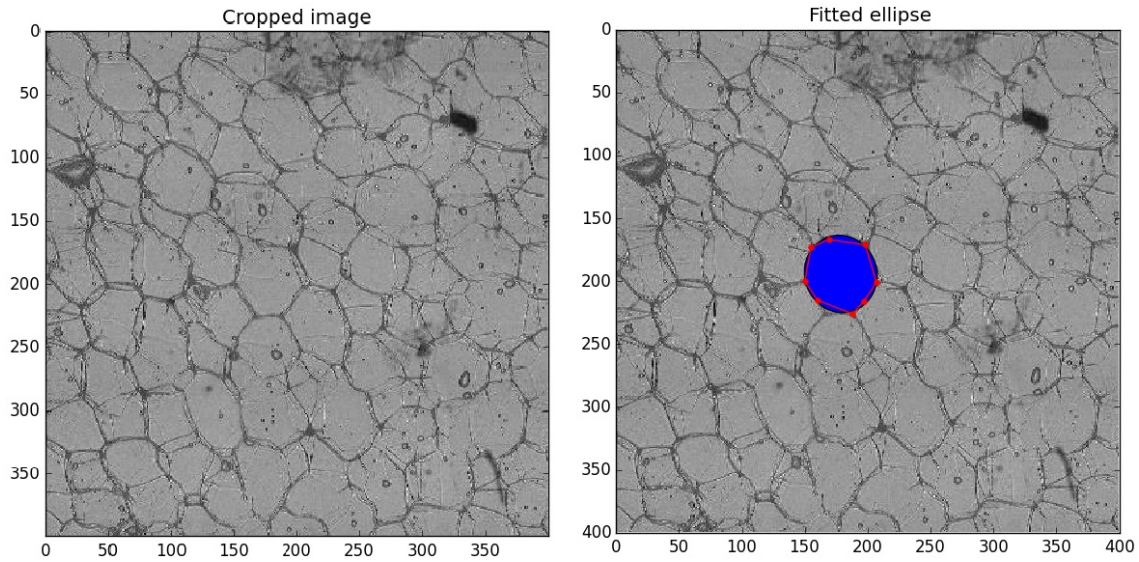


Figure 4.4: Again we see the cropped image on the left. In the right image a fitted ellipse is given, the particular cell is very round in shape so the fit is very nice. The vertices are found using a greedy snake algorithm, which will be described in Section 5.3.

The result looks nice (Fig. 4.4), but still we are not happy. We want to come really close to the actual cell shape, because we want to give a reliable result when we perform the analysis.

4.4. Edge enhancement

When we use the edges, we make a few further assumptions. First of all we need to limit the amount of edges, in particular since we add the most likely edges. We assume that the most likely connections are between points that are close together, which makes sense as all the vertices are located on the cell walls.

Since the image is quite grainy we also find the 'line-wise' local minima, i.e. the minima in the x, y or one of the diagonal directions. Instead of using a 3×3 ones mask we use a mask where there are only ones on a line through the central element. This results in minima (or maxima) in a given direction. So as for example we consider mask M_x

$$M_x = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}. \tag{4.8}$$

Using this particular mask we find the minima in the x direction. For the other three directions we use the other three straight lines that are possible. The improvement here is that most of the darker impurities get removed so they do not contribute to potential edges. In Figure 4.5 we see the minima that were found by the algorithm. Take note that the colour range of the lines is due to our summing the convolutions. This shows us that some of the dark lines are found in more that one direction. The algorithm that makes use of these lines only receives a boolean array in which the minima (i.e. the lines) are ones and the background are zeros. On this topic also see Section A.2.2.

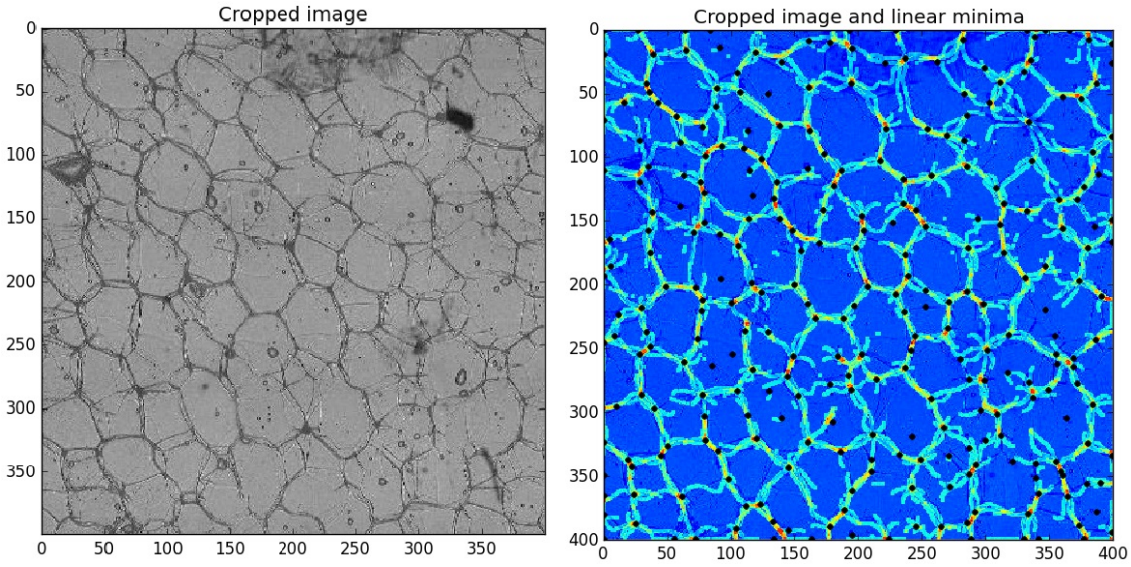


Figure 4.5: In the right panel we see the minima in each of the four directions. The lines are convolved with a 3×3 all ones mask, so in areas where the lines are very close they overlap and get a more red colour. Note how well that these lines follow the actual cell walls when we look at the real image in the left panel. The black dots are the located vertices.

4.4.1. Finding the right edges

For now let us first use the original image, and not the enhanced edges to find the cells. Basically what we want is to connect the vertices in such a way that the connecting lines lay on top of actual cell walls. In the original image the most obvious marker for cell walls is the darker colour. So an algorithm that makes use of this idea must compare the darkness in the direct neighbourhood of the edge.

This may be done in various ways, the simplest way is to add lines for which the underlying pixels are of a certain brightness. So the average pixel brightness is less than a given value. The first danger we spot is that when a cell wall is curved away from the straight line between two vertices, the average brightness will be too high and no connection will be drawn. In order to solve such a problem one can look at the pixels surrounding the line and see if they are dark enough. This means that the cell wall does not necessarily have to be on a straight line between two vertices. Note that another problem is that when there are many dark impurities in the image this may also lead to false identification of edges.

A new problem then arises when three vertices are almost on the same line. Then there are are three valid edges, but only two are needed. This means that for instance we do not allow multiple connections under a small angle. And between two conflicting lines we choose the shortest.

Now this all seems to be very straightforward, but bear in mind that based on our hexagonal assumption there are six vertices and edges per cell. This results in a computational problem, as this requires a lot of work to check for all sets of edges.

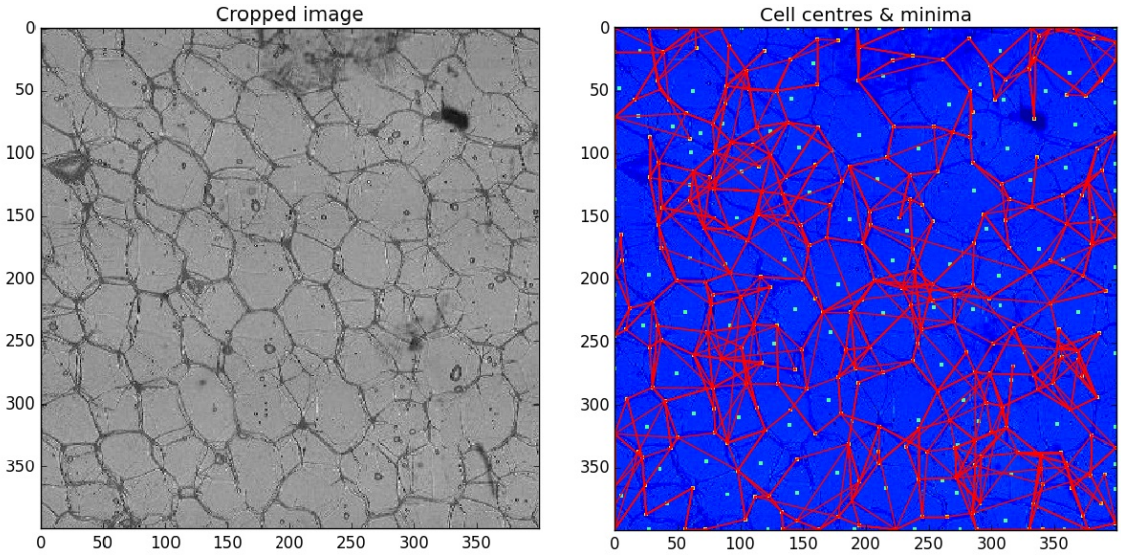


Figure 4.6: In the right image we see the located edges. For these edges we have used the image brightness. Some edges are correct, however a great part of the edges are wrong. The method has some promise, but it needs improvement.

A final problem that we look into is the problem that there can be many different valid edges for each vertex, so we must cap the number of allowed edges per vertex. This makes the analysis slow as we need to keep track of all the edges, not only the ones from a particular vertex, but also to each vertex. We see the result of the edge finding in Figure 4.6. Here the previously sketched problems, like small angles or adding too many connections to a vertex are visible. To improve the method we must impose extra conditions on the would-be edges.

So indeed we can find the edges, but the methods used here are very inefficient. Furthermore we note that even though we know the edges in the figure, we do not know to which cell they belong. In other words, an extra layer of analysis is needed to find the individual cells in the image.

4.4.2. Using located minima on lines

An extension to the edge finding algorithm then is that we can clear the image up by finding the minima in the four directions as described above. The resulting localisation is given in Figure 4.7. The edges are much more organised and we can already see how they follow the actual cell walls. Still there is the unpracticality of tracking down to which cell each edge belongs, so there is still much room for improvement.
These enhance edges still do not overcome all the other problems that we encounter using the edge finding algorithm, but it provides a more clean image for us to work with in other forms of analysis as we will see in Section 5.2.

4.5. Electrodynamics

Now we change the POI's for charges located at the POI's. We will find that we cannot use real mechanics, as we will find very poor cells. In fact a simpler method, which uses minima in the potential rather than forces on the charges, will return much better results. We want to find a stable solution, which gives a basic idea of what we want. From there we can go to better defined cells gradually. Finally we also examine if we can make the free charges attractive to one another.

4.5.1. Full dynamics

The force on a given charge Q from another charge q at some distance r is given by Coulomb's law (cf. *Introduction to electrodynamics* [23]):

$$\mathbf{F} = \frac{1}{4\pi\epsilon_0}\frac{qQ}{r^2}\mathbf{r}. \tag{4.9}$$

Now in the instance where there are multiple charges (k in total) they generate an electrical field $\mathbf{E}$,

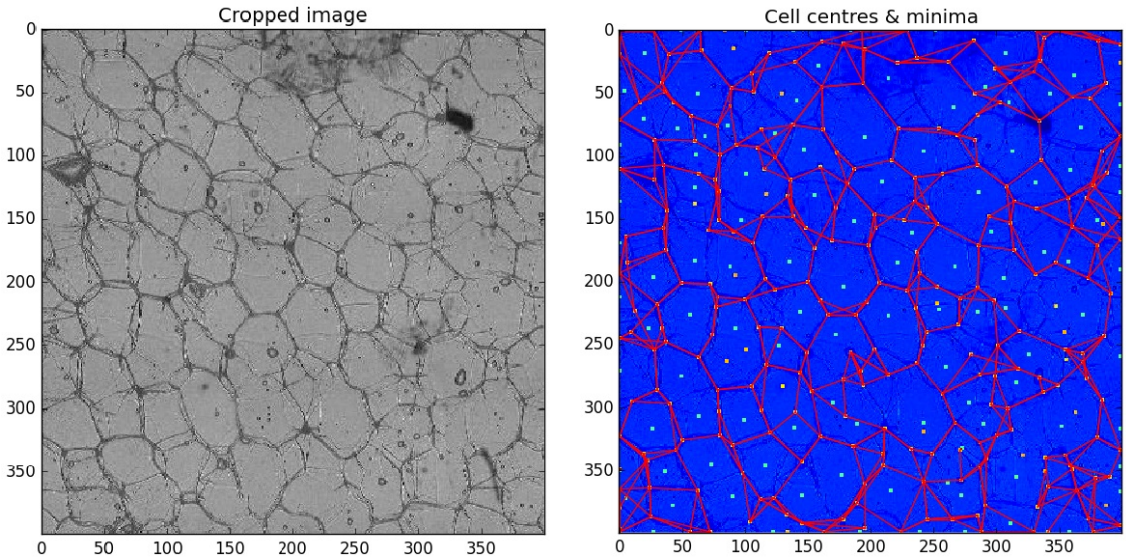


Figure 4.7: In the right image we see the located edges. In light blue the cells centres and pink the vertices, as we have found them. For the edges we have used the local minima in four directions as described above. The result is quite satisfying to look at, however the real data concerning the cells is still unknown. We need extra analysis to find out which edge belongs to which cell.

$$\mathbf{F} = Q\mathbf{E}, \text{ where } \mathbf{E} = \frac{1}{4\pi\epsilon_0}\sum_{i=1}^{k}\frac{q_i}{r_i^2}\mathbf{r}_i. \tag{4.10}$$

Let us consider the situation as we have it. There are three types of charges in the image, namely the located cell centres and vertices which are all fixed. The third group of charges are the movable charges that we have introduced to grow into the shape of the cell. We want the free charges to end in the vertices therefore they must be attracting the free charges. The cell centres however must repel the free charges. To prevent the free charges from merging into one point we either need to make the charge of the fixed points very large, or they must repel each other. For now we will go with repelling charges.

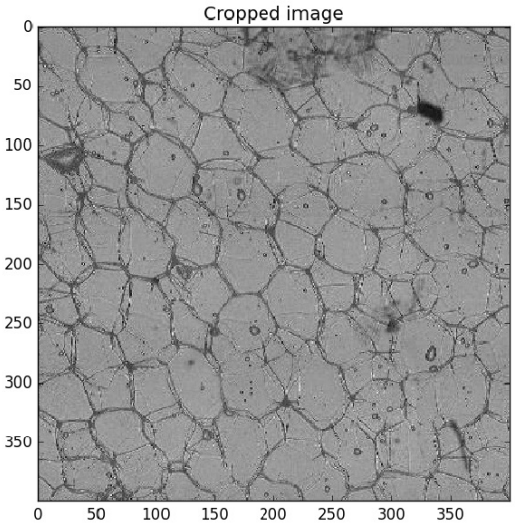
4.5.2. Stable solution

The final cell should be a stable solution, i.e. the force on the free charges should be zero. It is important to note that there is no analytical solution to a many body problem (which is essentially the problem we have). This also means that we can not calculate the exact stable points, as the free charges can move and thus change the force field. Therefore we can only solve the problem by simulation.

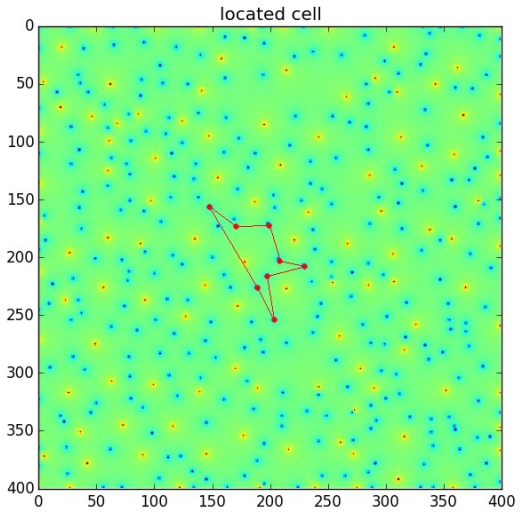
When simulating the movement of the charges we must bear in mind that the charges have a certain velocity and can therefore easily overshoot the vertices and move into the wrong positions. In order to prevent this we add a drag force $\mathbf{F}_{\text{drag}} = -\mu\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t}$.
When we examine the result in Figure 4.8 we see that even with added drag the cell has not been found properly. The best attempt is seen in Figure 4.8d, but even then the result is not good enough. It seems that using the full dynamics is not at all useful. We do not want to do away with the idea of attracting and repelling charges all together, but using the full dynamics complicates the computation greatly without providing the kind of result that we want.

4.5.3. Growing to shape

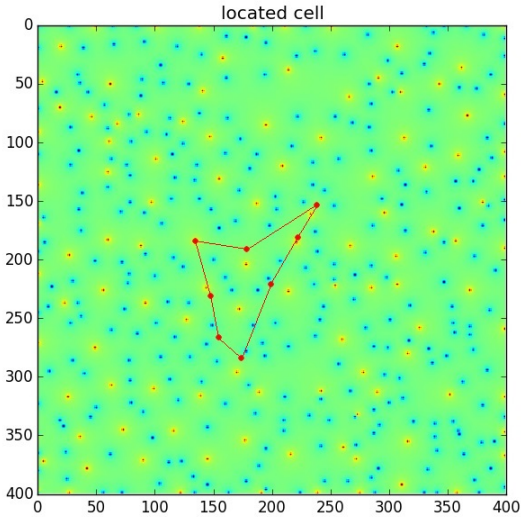
From what we now know we want to make the situation a less complicated. With the electrical field we also have a potential field, which we shall keep simple, and just take the sum of the potentials for each point charge. Here we have the potential of a point charge on the left and on the right the sum of multiple of these potentials:



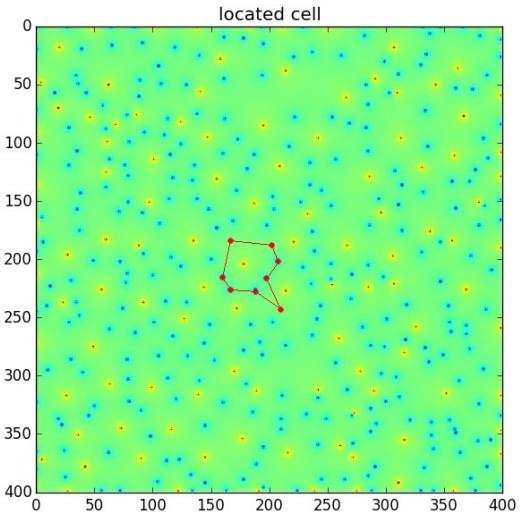
(a) Example image



(b) $Q = 1; q_c = 1; q_b = -1; \mu = 0.5$



(c) $Q = 0.8; q_c = 1; q_b = -1.5; \mu = 0$



(d) $Q = 0.8; qc = 1; qb = -1.5; \mu = 0.5$

Figure 4.8: We see three cells that were located using the full dynamics. Each time we try to find the same cell but we alter the parameters of the system. Where Q is the charge of the free particles, q_c, q_b the charges of the centres and vertices respectively and μ the drag coefficient. The cell that we want to find is the hexagonal cell in the centre of the image.

$$V(r) = \frac{1}{4\pi\epsilon_0} \sum_{i=1}^{k} \frac{q_i}{r_i}. \tag{4.11}$$

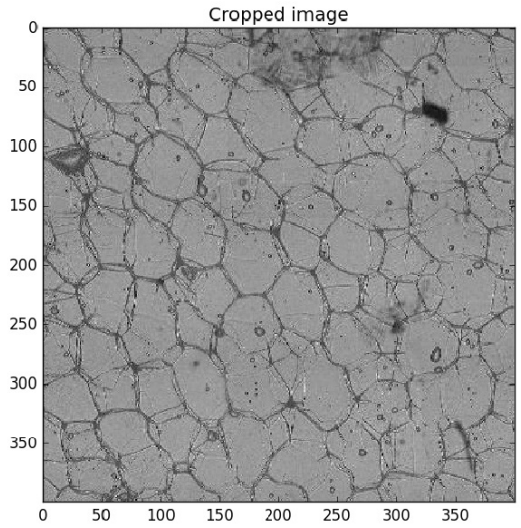
Then the solution is found by simply letting the free charges move into the local minima.

In order to let the charges grow into the full cell shape we have to come up with a simple algorithm. We start with eight free charges around the cell centre, then we look for the minimal potential in the direct neighbourhood of each charge, that is in a 3×3 square centred at the free charge. We allow this to continue until the charges stop moving. The algorithm in pseudocode becomes:

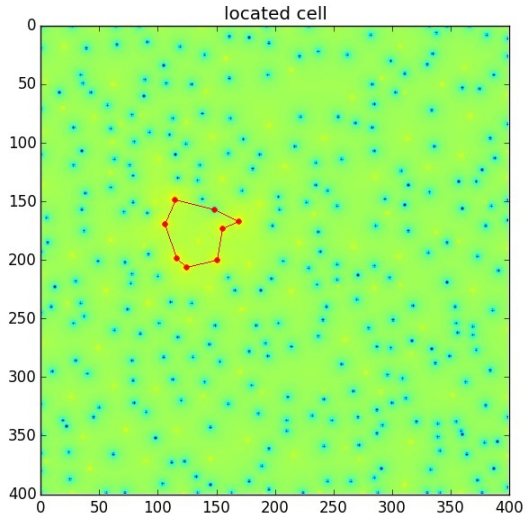
```
1. Initialise the free charges around a selected cell centre.

2. Calculate the potential of the fixed charges.

while difference != 0 or number < maximum:
```

```
for each free charge:
  3a. Calculate the potential of the free charges, add to total potential.
  3b. Find the minimal potential value near each free charge.
3c. Calculate difference with previous situation, stop when zero(/small).
3d. Check number of iterations, stop when maximum reached.
```

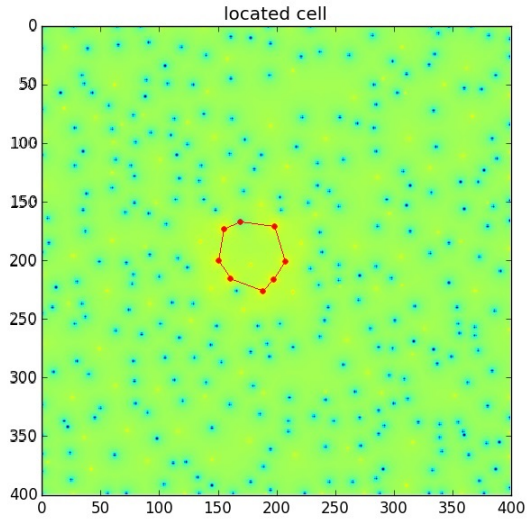
Listing 4.1: Pseudocode for finding the solution to the minimal potential.



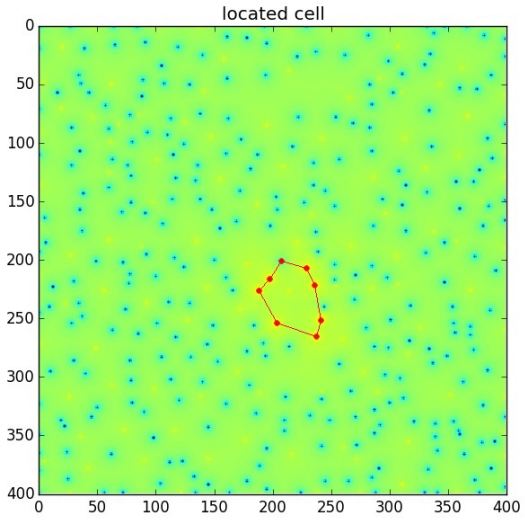
(a) Example image



(b) We see an error in the top right corner.



(c) The cell has 9 vertices so one could not be found.



(d) Again we miss out on one vertex, however comparing to the original image this fit is excellent.

Figure 4.9: We see three cells that were located using the potentials. In Figure 4.9c we see the same cell as we tried to find in Figure 4.8. It is clear that when we compare the results this looks much better than the full dynamics.

The result, shown in Figure 4.9, looks very pleasing for these individual cells, and we already know to which cell the located minima (i.e. vertices) belong so reconstruction is easy. Clearly there is something in this approach that we must hold on to. The problem is that it is very difficult to efficiently scale the algorithm up to the full image, because we need to keep track of all the free charges and look for the local minima around each free charge. Using this to find all the cells takes too much time.

So we want an algorithm that somehow makes use of attractors in the image, but it must also be easily scalable to find all the cells in a well ordered fashion. The structured organisation is important for further analysis. We combine these ideas in Chapter 5.

5

Snakes and GVF

5.1. Introduction

In this chapter we will demonstrate a final, and more formal approach, in which we use an active contour models. In particular we look at some algorithms concerning snakes. This can be done greedily and globally using a gradient vector model.

5.2. Active contours

In these notes we clarify the details around the active contour finding that we implement in order to find cells in the given images. That is to say, we look into the mathematical background, but also at the practical implementation thereof.

We start with a greedy algorithm to find the contours. For the contour we consider some closed parametric curve $\mathbf{r}(s) = \langle x(s), y(s) \rangle$, $s \in [a, b]$ (cf. [6, 7]). This contour evolves on an image of some intensity $\rho(x, y)$, in this evolution the contour tries to minimise an energy functional:

$$E(\mathbf{r}) = \int_a^b \left[\alpha(s) E^{\text{cnt}}(s) + \beta(s) E^{\text{crv}}(s) + \gamma(s) E^{\text{img}}(s) \right] \mathrm{d}s. \tag{5.1}$$

Here E^{cnt} forces a the contour to be continuous, E^{crv} enforces smoothness and E^{img} attracts to an image edge (i.e. an actual contour).

5.2.1. Continuity term

The continuity term is about minimising the first derivative:

$$E^{\text{cnt}}(s) = \left|\left| \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}s} \right|\right|^2. \tag{5.2}$$

In the discrete instance, the contour will be approximated by a certain number N points, $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N$. Then we can approximate the first derivative by a finite difference:

$$E^{\text{cnt}}(s) = ||\mathbf{p}_i - \mathbf{p}_{i-1}||^2 \Leftrightarrow E^{\text{cnt}}(s) = (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 \tag{5.3}$$

Note that this tries to minimise the difference between the points of the contour. This in turn means that it will have the effect of shrinking the contour, which is not always desirable. Therefore a more sophisticated form can be used

$$E^{\text{cnt}}(s) = \left(\overline{d} - ||\mathbf{p}_i - \mathbf{p}_{i-1}|| \right)^2, \tag{5.4}$$

here $\overline{d}$ is the average distance between the points. This structure has the effect of keeping the points at an equal distance along the curve, hence continue, without the strong shrinking effect.

5.2.2. Smoothing term

For the smoothness we minimise the second derivative, this term will add extra cost to strong curvatures. This in turn will avoid sharp corners and oscillations in the contour shape. So we minimise:

$$E^{\text{crv}}(s) = \left|\left|\frac{\mathrm{d}^2\mathbf{r}}{\mathrm{d}s^2}\right|\right|^2. \tag{5.5}$$

Again we look at the discrete case, and like before we approximate using a finite difference:

$$E^{\text{cnt}}(s) = ||\mathbf{p}_{i-1} - 2\mathbf{p}_i + \mathbf{p}_{i+1}||^2 \Leftrightarrow E^{\text{cnt}}(s) = (x_{i-1} - 2x_i + x_{i+1})^2 + (y_{i-1} - 2y_i + y_{i+1})^2 \tag{5.6}$$

In practice a strong smoothness term will prevent the contours from growing, and in turn will have a shrinking effect.

Here we should note that we invert the process, as we start with a contour smaller than the object that we want to find. Whereas the actual algorithm is supposed to shrink around a particular contour. To visualise, we are inflating a balloon in a certain constricted area. The normal usage of the algorithm is to vacuum seal a certain object with a contour.

5.2.3. Image term

The final term is the image based term. In the image the contours that need to be located are made to attract the curve $\mathbf{r}(s)$. This can be achieved in different ways, we choose to preprocess the image such that we get a smooth landscape $\rho_{pre}(x,y)$. In this landscape cell centres that we have located repel the contour and the located cell boundaries attract the contour. In our case we get:

$$E^{\text{img}}(s) = \rho_{pre}(x,y). \tag{5.7}$$

In our case this image energy becomes negative (or at least small) near the located cell walls. Another way to use this is by taking the gradient of the image intensity $E^{\text{img}}(s) = -||\nabla I||$, then near edges the image energy becomes very small.

5.2.4. Discretisation

As we have noted above we can use a discrete setup rather than a continuous equation. This works very well as the image consists of discrete pixels. We then also need to assume a limited discrete number of points on the parametrised curve that we use.
Now we are left to minimise a sum:

$$E(\mathbf{r}) = \sum_{i=1}^{N} \left[\alpha(s_i)E^{\text{cnt}}(s_i) + \beta(s_i)E^{\text{crv}}(s_i) + \gamma(s_i)E^{\text{img}}(s_i)\right]. \tag{5.8}$$

5.3. Greedy algorithm

Now instead of minimising the sum of Eq. (5.8) we choose to find a local minimum. This implies that around each point p_i on the curve we find a local minimum in a small area around it of some size $M \times M$ and move the point there. We repeat this process until the points are stable. This also means that we actually only need the location of three points for each calculation. We need the current location of point $\mathbf{p}_i$ and the location of its neighbours $\mathbf{p}_{i-1}, \mathbf{p}_{i+1}$. Substituting this information into the various discrete formulations that we have, we need to minimise over some small area each time.

We can add alterations to enhance the performance of the algorithm, for known impurities of the image we can set the value of $\gamma(s_i)$ to zero. However this involves a lot of meddling with the image which for now we do not want. Instead the algorithm needs to works as autonomously as possible (as we need to detect some 2500 contours in an image).

We implement one additional step, we do not want the snake to grow tails along the edges of neighbouring cells, therefore if two points $\mathbf{p}_j$ and $\mathbf{p}_{j+2}$ are too close we remove point $\mathbf{p}_{j+1}$ as it is obviously making a very sharp corner which is presumably becoming a tail in the long run. In addition this helps to prevent the contour from getting sharp corners, which is desirable as we assume that cells are more or less smooth convex shapes.

This method returns very good results (Fig. 5.1), most cells are found. Note that even here the impurities, in particular in Figure 5.1b, have a strong influence. However the total result is very nice, not only is it graphically satisfying, but it is also practical as we have a structured list of cells. The main downside is that it still lacks speed.

5.4. Gradient vector flow and active contour

The greedy algorithm provides good results, this suggests that we investigate the actual solving of the minimisation of the energy functional as in Eq. (5.1). Using the results found in [10], we write the energy functional

$$E(\mathbf{r}(s)) = \int_0^1 \frac{1}{2}(\alpha|\mathbf{r}'(s)|^2 + \beta|\mathbf{r}''(s)|^2) + \gamma E^{\text{img}}(s)\mathrm{d}s. \tag{5.9}$$

To minimise the functional (5.9) over all closed curves $\mathbf{r}(s)$ we consider the variational formulation, as in Section A.3:

$$\frac{d}{d\epsilon}E(\mathbf{r}+\epsilon\mathbf{v})\bigg|_{\epsilon=0} = 0, \quad \forall\mathbf{v} \tag{5.10}$$

which leads to the Euler-Lagrange equation:

$$\begin{aligned} \alpha\mathbf{r}''(s) - \beta\mathbf{r}''''(s) - \gamma\nabla E^{\text{img}}\big|_{(x,y)\in\mathbf{r}(s)} = \mathbf{0}, \quad 0 < s < 1 \\ \mathbf{r}(0) = \mathbf{r}(1), \quad \mathbf{r}'(0) = \mathbf{r}'(1), \quad \mathbf{r}''(0) = \mathbf{r}''(1), \quad \mathbf{r}'''(0) = \mathbf{r}'''(1) \end{aligned} \tag{5.11}$$

To satisfy this equation the curve may be evolved in time ($t \geq 0$) as

$$\begin{aligned} &\frac{\partial\mathbf{r}(s,t)}{\partial t} = \alpha\frac{\partial^2\mathbf{r}(s,t)}{\partial s^2} - \beta\frac{\partial^4\mathbf{r}(s,t)}{\partial s^4} - \gamma\nabla E^{\text{img}}\big|_{(x,y)\in\mathbf{r}(s,t)}, \quad 0 < s < 1, \\ &\mathbf{r}(s,0) = \mathbf{r}_0(s), \quad 0 \leq s \leq 1, \\ &\mathbf{r}(0,t) = \mathbf{r}(1,t), \quad \frac{\partial\mathbf{r}(0,t)}{\partial s} = \frac{\partial\mathbf{r}(1,t)}{\partial s}, \\ &\frac{\partial^2\mathbf{r}(0,t)}{\partial s^2} = \frac{\partial^2\mathbf{r}(1,t)}{\partial s^2}, \quad \frac{\partial^3\mathbf{r}(0,t)}{\partial s^3} = \frac{\partial^3\mathbf{r}(1,t)}{\partial s^3}, \end{aligned} \tag{5.12}$$

until it reaches a steady state, which is equivalent to the equation of motion under the action of internal and external forces:

$$\frac{\partial\mathbf{r}(s,t)}{\partial t} = \mathbf{F}^{\text{int}}(\mathbf{r}(s,t)) + \mathbf{F}^{\text{ext}}(\mathbf{r}(s,t)), \tag{5.13}$$

where the internal force is:

$$\mathbf{F}^{\text{int}}(\mathbf{r}(s,t)) = \alpha\frac{\partial^2\mathbf{r}(s,t)}{\partial s^2} - \beta\frac{\partial^4\mathbf{r}(s,t)}{\partial s^4}, \tag{5.14}$$

and the edge-attracting external force is:

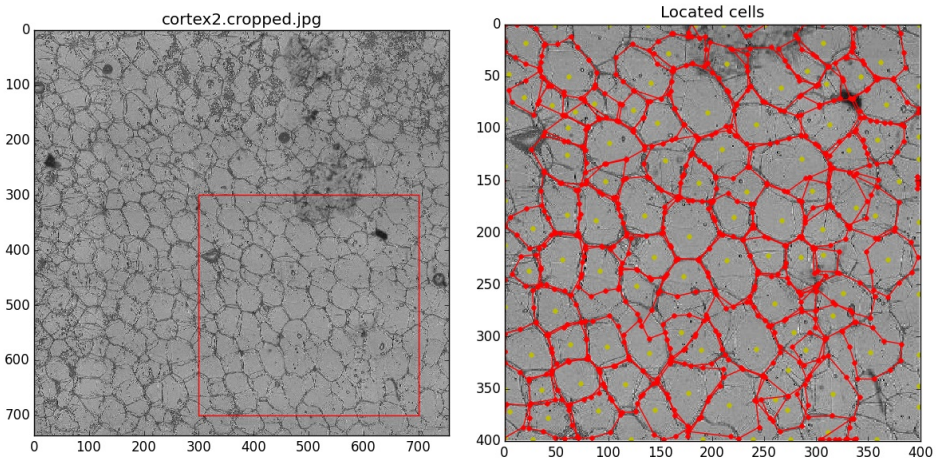
$$\mathbf{F}^{\text{ext}}(\mathbf{r}(s,t)) = -\gamma\nabla E^{\text{img}}\big|_{(x,y)\in\mathbf{r}(s,t)}. \tag{5.15}$$

Other forces may be included as well.

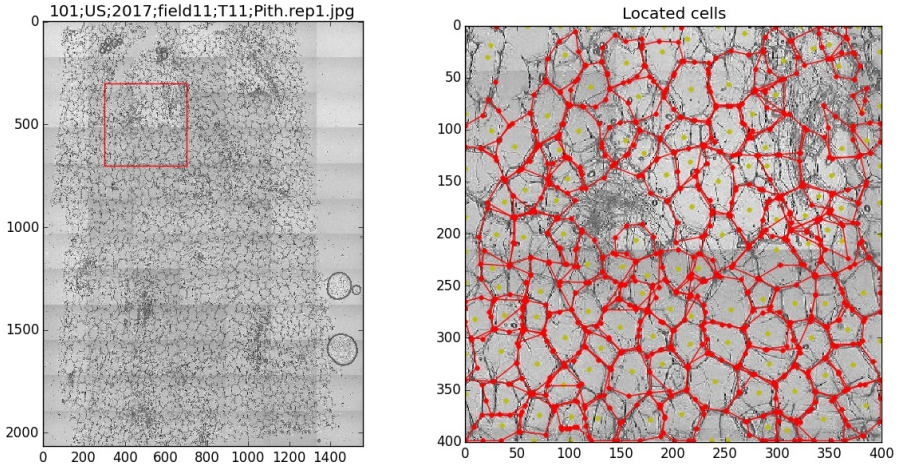
5.4.1. The process

The dynamic Equation (5.12)–(5.13) will successfully evolve the curve towards an edge starting from some initial guess, provided the external driving force $\mathbf{F}^{\text{ext}}$ has a sufficiently long range. In practice this usually means placing the initial contour close to the target edge. To obtain convergent behaviour for any initial guess, instead of $\mathbf{F}^{\text{ext}}$ given by (5.15), one can introduce a long-range vector field that guides the contour towards the nearby edge from anywhere inside the image. This vector field $\mathbf{w}(x,y) = \langle u(x,y), v(x,y)\rangle$ can be constructed as a minimizer of the following functional over the image domain Ω:

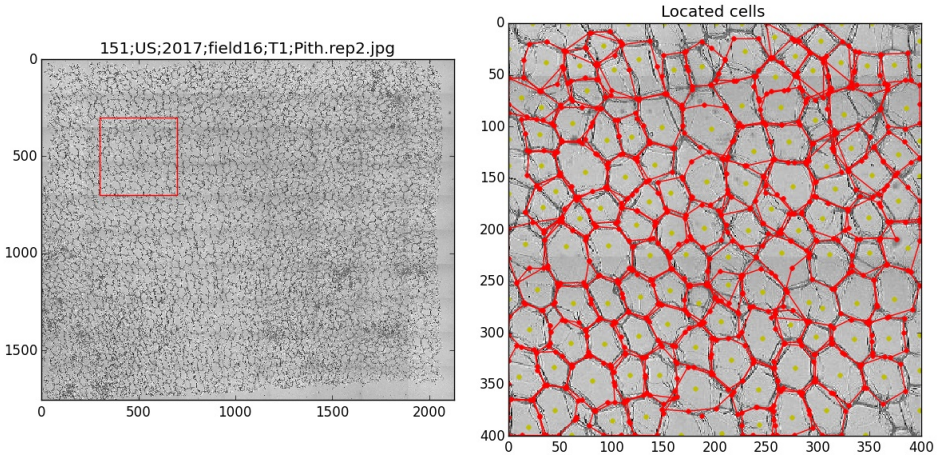
$$\mathscr{E}(\mathbf{w}) = \iint_\Omega \left[\mu\left(|\nabla u|^2 + |\nabla v|^2\right) + \left|\nabla E^{\text{img}}\right|^2\left|\mathbf{w} - \nabla E^{\text{img}}\right|^2\right] dA, \tag{5.16}$$



(a) In the red square we see the area on which the greedy snake algorithm is implemented, in the right panel we have the result. The original image can be seen in Appendix B, in Figure B.1.



(b) We use the same greedy snake algorithm, but now on a different larger image. Original Figure B.2.



(c) Yet another image on which we allow the greedy snake algorithm to operate. Original Figure B.3.

Figure 5.1: Three example segmentations found by the greedy snake algorithm. Notice how well most cell contours are found using this algorithm. We see that the impurities still interfere with the result, in particular in Figure 5.1b.

where μ is the regularisation parameter. The corresponding Euler equations are:

$$\begin{aligned}
&\mu\Delta u - \left(u - \frac{\partial E^{\text{img}}}{\partial x}\right)\left[\left(\frac{\partial E^{\text{img}}}{\partial x}\right)^2 + \left(\frac{\partial E^{\text{img}}}{\partial y}\right)^2\right] = 0,\\
&\mu\Delta v - \left(v - \frac{\partial E^{\text{img}}}{\partial y}\right)\left[\left(\frac{\partial E^{\text{img}}}{\partial x}\right)^2 + \left(\frac{\partial E^{\text{img}}}{\partial y}\right)^2\right] = 0.\\
&\nabla u \cdot \mathbf{n}|_{\mathbf{x}\in\partial\Omega} = 0, \quad \nabla v \cdot \mathbf{n}|_{\mathbf{x}\in\partial\Omega} = 0,
\end{aligned} \tag{5.17}$$

where $\mathbf{n}$ is the outward unit normal vector of $\partial\Omega$. These are screened Poisson's equations, see for example *Innovations for shape analysis: models and algorithms* [26, pp. 453].

Having found $\mathbf{w} = \langle u, v\rangle$ from (5.17), the edge-conforming contour is obtained as a steady state solution of

$$\frac{\partial \mathbf{r}(s,t)}{\partial t} = \mathbf{F}^{\text{int}}(\mathbf{r}(s,t)) + \mathbf{w}(\mathbf{r}(s,t)) \tag{5.18}$$

with $\mathbf{F}^{\text{int}}$ given by (5.14).

5.4.2. Discretised GVF equations

We apply the Finite Difference Method (FDM) to discretise the Gradient Vector Flow (GVF) equations (5.17). First, GVF equations are re-written in the form:

$$\begin{aligned} \mu\Delta u - ku &= f, \quad (x,y) \in \Omega; \\ \mu\Delta v - kv &= g, \quad (x,y) \in \Omega; \\ \nabla u \cdot \mathbf{n} = \nabla v \cdot \mathbf{n} &= 0, \quad (x,y) \in \partial\Omega \end{aligned} \tag{5.19}$$

where

$$\begin{aligned} k(x,y) &= \left(\frac{\partial E^{\text{img}}}{\partial x}\right)^2 + \left(\frac{\partial E^{\text{img}}}{\partial y}\right)^2 = \left(\nabla E^{\text{img}}\right) \cdot \left(\nabla E^{\text{img}}\right), \\ f(x,y) &= -k\frac{\partial E^{\text{img}}}{\partial x}, \\ g(x,y) &= -k\frac{\partial E^{\text{img}}}{\partial y}. \end{aligned} \tag{5.20}$$

If the image energy density E^{img} is given by:

$$E^{\text{img}}(x,y) = -\sqrt{(\nabla\rho)\cdot(\nabla\rho)}, \tag{5.21}$$

then its computation requires some additional work. Introducing the notation $\rho_{i,j} = \rho(x_i, y_j)$ and taking into account that the coordinates (x_i, y_i) of the image pixels have integer values $x_i = 0, 1, \ldots, n+1$; $y_j = 0, 1, \ldots, m+1$, the central-difference approximations of the first derivatives are:

$$\begin{aligned} \left.\frac{\partial\rho(x,y)}{\partial x}\right|_{(x,y)=(x_i,y_j)\in\Omega} &\approx \frac{1}{2}\left(\rho_{i+1,j} - \rho_{i-1,j}\right), \quad i = 1,\ldots,n; \;\; j = 1,\ldots,m; \\ \left.\frac{\partial\rho(x,y)}{\partial y}\right|_{(x,y)=(x_i,y_j)\in\Omega} &\approx \frac{1}{2}\left(\rho_{i,j+1} - \rho_{i,j-1}\right), \quad i = 1,\ldots,n; \;\; j = 1,\ldots,m; \end{aligned} \tag{5.22}$$

These operations can be expressed as sparse matrix-vector multiplications. To this end we introduce the following FD matrix $D_n \in \mathbb{R}^{n\times(n+2)}$ corresponding to the first derivative:

$$D_n = \begin{bmatrix} -1 & 0 & 1 & 0 & \ldots & & \\ 0 & -1 & 0 & 1 & \ldots & & \\ & & & \ddots & & & \\ & & & & \ddots & & \\ & & & & & \ddots & \\ & & & & \ldots & -1 \;\; 0 \;\; 1 & \end{bmatrix} \tag{5.23}$$

as well as its extended version $\hat{D}_{n+2} \in \mathbb{R}^{(n+2)\times(n+2)}$ with one-sided FD derivatives at the image boundaries:

$$\hat{D}_{n+2} = \begin{bmatrix} -1 & 1 & 0 & 0 & \dots & & & \\ -1 & 0 & 1 & 0 & \dots & & & \\ 0 & -1 & 0 & 1 & \dots & & & \\ & & & \ddots & & & & \\ & & & & \ddots & & & \\ & & & & & \ddots & & \\ & & & & \dots & -1 & 0 & 1 \\ & & & & \dots & 0 & -1 & 1 \end{bmatrix} \tag{5.24}$$

We sort the image $\rho_{i,j}$ data in lexicographic order and store it in the vector $\boldsymbol{\rho} \in \mathbb{R}^{(n+2)(m+2)}$ as:

$$\boldsymbol{\rho} = [\rho_{0,0}\, \rho_{1,0} \dots \rho_{n+1,0}\, \rho_{0,1}\, \rho_{1,1} \dots\dots \rho_{n+1,m+1}]^T. \tag{5.25}$$

In this case the required FD representations $\mathbf{k}, \mathbf{f}, \mathbf{g} \in \mathbb{R}^{nm}$ of $k(x,y)$, $f(x,y)$ and $g(x,y)$ can be constructed with the help of Kronecker products and entry wise operations. First, we define several FD matrices:

$$\begin{aligned} \hat{D}_x &= \frac{1}{2}\left(I_{m+2} \otimes \hat{D}_{n+2}\right), \\ \hat{D}_y &= \frac{1}{2}\left(\hat{D}_{m+2} \otimes I_{n+2}\right), \\ D_x &= \frac{1}{2}\left(I_m \otimes D_n\right), \\ D_y &= \frac{1}{2}\left(D_m \otimes I_n\right). \end{aligned} \tag{5.26}$$

Then, for the image energy defined as in Equation (5.7), the discretised vectors $\mathbf{k}$, $\mathbf{f}$, and $\mathbf{g}$ are obtained as follows:

$$\begin{aligned} E^{\text{img}}(x,y) \to \mathbf{e}^{\text{img}} &= -\sqrt{|\hat{D}_x\boldsymbol{\rho}|^2 + |\hat{D}_y\boldsymbol{\rho}|^2}, \\ k(x,y) \to \mathbf{k} &= |D_x\mathbf{e}^{\text{img}}|^2 + |D_y\mathbf{e}^{\text{img}}|^2, \\ f(x,y) \to \mathbf{f} &= -\mathbf{k} \circ \left(D_x\mathbf{e}^{\text{img}}\right), \\ g(x,y) \to \mathbf{g} &= -\mathbf{k} \circ \left(D_y\mathbf{e}^{\text{img}}\right), \end{aligned} \tag{5.27}$$

where I_n is the $n \times n$ identity matrix; the absolute value, the power, and the square root are entry wise operations; and $\mathbf{a} \circ \mathbf{b}$ denotes the entry wise (Hadamard) product of two vectors of the same size.

If the image energy is defined as in (5.21), then the required vectors are obtained as:

$$\begin{aligned} k(x,y) \to \mathbf{k} &= |D_x\boldsymbol{\rho}|^2 + |D_y\boldsymbol{\rho}|^2, \\ f(x,y) \to \mathbf{f} &= -\mathbf{k} \circ \left(D_x\boldsymbol{\rho}\right), \\ g(x,y) \to \mathbf{g} &= -\mathbf{k} \circ \left(D_y\boldsymbol{\rho}\right), \end{aligned} \tag{5.28}$$

With the help of the 1D backward-derivative FD operator $\tilde{D}_n \in \mathbb{R}^{(n-1)\times n}$:

$$\tilde{D}_n = \begin{bmatrix} -1 & 1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 1 & 0 & \dots & \vdots \\ \vdots & & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \vdots \\ 0 & \dots & & 0 & -1 & 1 \end{bmatrix}, \tag{5.29}$$

the action of 2D Laplacian on u can be expressed in the FD form as:

$$\Delta u \to -L\mathbf{u} = -\left[I_m \otimes \left(\tilde{D}_n^T\tilde{D}_n\right) + \left(\tilde{D}_m^T\tilde{D}_m\right) \otimes I_n\right]\mathbf{u}, \tag{5.30}$$

and similarly for $\Delta v \to -L\mathbf{v}$. Here $\mathbf{u},\mathbf{v} \in \mathbb{R}^{nm}$. Finally, we arrive at the following linear algebraic problems:

$$\begin{aligned} \big[\mu L + \mathrm{diag}(\mathbf{k})\big]\,\mathbf{u} &= -\mathbf{f}, \\ \big[\mu L + \mathrm{diag}(\mathbf{k})\big]\,\mathbf{v} &= -\mathbf{g}, \end{aligned} \tag{5.31}$$

where diag($\mathbf{k}$) is the diagonal matrix with the entries of vector $\mathbf{k}$ on its diagonal.

5.4.3. AC equation

To discretise the active contour (AC) equation (Eq. (5.18)) we represent each curve at time t_k as a collection of p points: $\mathbf{r}_{q,k} = \langle x_q(t_k), y_q(t_k)\rangle$, $q = 1,\ldots,p$; consecutively spread over the active contour. An m-th order curve has the general equation:

$$\mathbf{r}(s) = \mathbf{r}(0) + \mathbf{r}^{(1)}(0)\,s + \frac{1}{2}\mathbf{r}^{(2)}(0)\,s^2 + \cdots + \frac{1}{m}\mathbf{r}^{(m)}(0)\,s^m. \tag{5.32}$$

Hence, it is sufficient to known the coordinates of $m+1$ points along the curve to find an approximation for the first m derivatives of $\mathbf{r}(s)$ at $s = 0$.

Let the curve parameter take three discrete values $s = -1, 0, 1$ at the points $\mathbf{r}_{q-1,k}$, $\mathbf{r}_{q,k}$, and $\mathbf{r}_{q+1,k}$. Then, retaining in (5.32) terms up to s^2, we arrive at the following three equations:

$$\begin{aligned} \mathbf{r}_{q-1,k} &= \mathbf{r}_{q,k} - \mathbf{r}^{(1)}_{q,k} + \frac{1}{2}\mathbf{r}^{(2)}_{q,k}, \\ \mathbf{r}_{q,k} &= \mathbf{r}_{q,k}, \\ \mathbf{r}_{q+1,k} &= \mathbf{r}_{q,k} + \mathbf{r}^{(1)}_{q,k} + \frac{1}{2}\mathbf{r}^{(2)}_{q,k}. \end{aligned} \tag{5.33}$$

Adding the first and the last equation we obtain the FD approximation of the second derivative:

$$\left.\frac{\partial^2 \mathbf{r}(s,t)}{\partial s^2}\right|_{\mathbf{r}(s,t)=\mathbf{r}_{q,k}} \approx \mathbf{r}^{(2)}_{q,k} = \mathbf{r}_{q-1,k} - 2\mathbf{r}_{q,k} + \mathbf{r}_{q+1,k}. \tag{5.34}$$

Similarly, to approximate the fourth derivative $\mathbf{r}^{(4)}(s)$ we consider terms up to s^4 and five points along the curve corresponding to $s = -2, -1, 0, 1, 2$:

$$\begin{aligned} \mathbf{r}_{q-2,k} &= \mathbf{r}_{q,k} - 2\mathbf{r}^{(1)}_{q,k} + 2\mathbf{r}^{(2)}_{q,k} - \frac{4}{3}\mathbf{r}^{(3)}_{q,k} + \frac{2}{3}\mathbf{r}^{(4)}_{q,k}, \\ \mathbf{r}_{q-1,k} &= \mathbf{r}_{q,k} - \mathbf{r}^{(1)}_{q,k} + \frac{1}{2}\mathbf{r}^{(2)}_{q,k} - \frac{1}{6}\mathbf{r}^{(3)}_{q,k} + \frac{1}{24}\mathbf{r}^{(4)}_{q,k}, \\ \mathbf{r}_{q,k} &= \mathbf{r}_{q,k}, \\ \mathbf{r}_{q+1,k} &= \mathbf{r}_{q,k} + \mathbf{r}^{(1)}_{q,k} + \frac{1}{2}\mathbf{r}^{(2)}_{q,k} + \frac{1}{6}\mathbf{r}^{(3)}_{q,k} + \frac{1}{24}\mathbf{r}^{(4)}_{q,k}, \\ \mathbf{r}_{q+2,k} &= \mathbf{r}_{q,k} + 2\mathbf{r}^{(1)}_{q,k} + 2\mathbf{r}^{(2)}_{q,k} + \frac{4}{3}\mathbf{r}^{(3)}_{q,k} + \frac{2}{3}\mathbf{r}^{(4)}_{q,k}. \end{aligned} \tag{5.35}$$

Adding equations we get:

$$\begin{aligned} \mathbf{r}_{q-2,k} + \mathbf{r}_{q+2,k} &= 2\mathbf{r}_{q,k} + 4\mathbf{r}^{(2)}_{q,k} + \frac{4}{3}\mathbf{r}^{(4)}_{q,k}, \\ \mathbf{r}_{q-1,k} + \mathbf{r}_{q+1,k} &= 2\mathbf{r}_{q,k} + \mathbf{r}^{(2)}_{q,k} + \frac{1}{12}\mathbf{r}^{(4)}_{q,k}. \end{aligned} \tag{5.36}$$

Multiplying the second equation by 4 and subtracting it from the first equation we obtain the FD approximation of the fourth derivative:

$$\left.\frac{\partial^4 \mathbf{r}(s,t)}{\partial s^4}\right|_{\mathbf{r}(s,t)=\mathbf{r}_{q,k}} \approx \mathbf{r}^{(4)}_{q,k} = \mathbf{r}_{q-2,k} - 4\mathbf{r}_{q-1,k} + 6\mathbf{r}_{q,k} - 4\mathbf{r}_{q+1,k} + \mathbf{r}_{q+2,k}. \tag{5.37}$$

Further, we store the points of the active contour $\mathbf{r}_{q,k} = \langle x_{q,k}, y_{q,k}\rangle$, $q = 1,\ldots,p$ in two vectors $\mathbf{x}_k, \mathbf{y}_k \in \mathbb{R}^{2p}$ as follows:

$$\begin{aligned} \mathbf{x}_k &= [x_{1,k}\; x_{2,k} \ldots x_{p,k}]^T \\ \mathbf{y}_k &= [y_{1,k}\; y_{2,k} \ldots y_{p,k}]^T \end{aligned} \tag{5.38}$$

Then, taking into account periodicity of points over a closed contour ($\mathbf{r}_{p+1,k} = \mathbf{r}_{1,k}$), the simultaneous action of the second derivative on all points of the contour can be represented as the matrix-vector product $D^{(2)}\mathbf{x}_k$ with the matrix $D_s^{(2)} \in \mathbb{R}^{p\times p}$ having the following structure:

$$D_s^{(2)} = \begin{bmatrix} -2 & 1 & 0 & \dots & 0 & 1 \\ 1 & -2 & 1 & 0 & \dots & 0 \\ & & & \ddots & & \\ 0 & \dots & 0 & 1 & -2 & 1 \\ 1 & 0 & \dots & 0 & 1 & -2 \end{bmatrix}. \tag{5.39}$$

The action of the fourth derivative is given by the matrix $D_s^{(4)} \in \mathbb{R}^{p\times p}$:

$$D_s^{(4)} = \begin{bmatrix} 6 & -4 & 1 & 0 & \dots & & 0 & 1 & -4 \\ -4 & 6 & -4 & 1 & 0 & \dots & & 0 & 1 \\ 1 & -4 & 6 & -4 & 1 & 0 & \dots & & 0 \\ & & & & \ddots & & & & \\ 0 & & \dots & 0 & 1 & -4 & 6 & -4 & 1 \\ 1 & 0 & & \dots & 0 & 1 & -4 & 6 & -4 \\ -4 & 1 & 0 & & \dots & 0 & 1 & -4 & 6 \end{bmatrix}. \tag{5.40}$$

Applying the forward Euler method for time integration, the discretised AC equation becomes

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \Delta t\left(\alpha D_s^{(2)} - \beta D_s^{(4)}\right)\mathbf{x}_k + \Delta t\mathbf{u}_k, \\ \mathbf{y}_{k+1} &= \mathbf{y}_k + \Delta t\left(\alpha D_s^{(2)} - \beta D_s^{(4)}\right)\mathbf{y}_k + \Delta t\mathbf{v}_k, \end{aligned} \tag{5.41}$$

where $\mathbf{u}_k, \mathbf{v}_k \in \mathbb{R}^p$ contain the p elements of $\mathbf{u}$ and $\mathbf{v}$ corresponding to the current location of the contour points. To ensure stability for larger time steps we can also use the Euler backward method. Let us use a hybrid form and use the backward Euler method only on the internal snake terms, but not on the GVF forces.

$$\begin{aligned} \left(I - \Delta t\left(\alpha D_s^{(2)} - \beta D_s^{(4)}\right)\right)\mathbf{x}_{k+1} &= \mathbf{x}_k + \Delta t\mathbf{u}_k, \\ \left(I - \Delta t\left(\alpha D_s^{(2)} - \beta D_s^{(4)}\right)\right)\mathbf{y}_{k+1} &= \mathbf{y}_k + \Delta t\mathbf{v}_k. \end{aligned} \tag{5.42}$$

Here I represents the proper sized identity matrix.

5.4.4. GVF parameters

Now we have a set of iterative functions, we still want to understand how the values of the parameters change the outcome. We know that α goes with the continuity term, and β controls the curvature. Finally γ controls the image force. In order to understand the equation better we look at Equation (5.13). Let us rewrite it here with all the terms included.

$$\frac{\partial \mathbf{r}(s,t)}{\partial t} = \alpha\frac{\partial^2 \mathbf{r}(s,t)}{\partial s^2} - \beta\frac{\partial^4 \mathbf{r}(s,t)}{\partial s^4} + \mathbf{F}^{\text{ext}}(\mathbf{r}(s,t)). \tag{5.43}$$

Now to solve this PDE we can set $\beta, \gamma = 0$, then we have a simple second derivative. Remember that we have periodic boundary conditions.

$$\frac{\partial \mathbf{r}(s,t)}{\partial t} = \alpha\frac{\partial^2 \mathbf{r}(s,t)}{\partial s^2}. \tag{5.44}$$

For the second derivative operator we know the eigenvalues and eigenfunctions,

$$\begin{aligned} &\frac{\partial^2 \mathbf{v}_k}{\partial s^2} = \lambda_k \mathbf{v}_k, \quad \lambda_k \le 0 \ \forall k \in \{1,2,\dots\}. \\ &\lambda_k = \begin{cases} -\frac{k^2\pi^2}{L^2} & k \text{ is even.} \\ -\frac{(k-1)^2\pi^2}{L^2} & k \text{ is odd.} \end{cases}, \quad \mathbf{v}_k = \begin{cases} L^{-1/2} & j = 1 \\ -\sqrt{\frac{2}{L}}\sin\left(\frac{k\pi x}{L}\right) & k \text{ is even.} \\ -\sqrt{\frac{2}{L}}\cos\left(\frac{(k-1)\pi x}{L}\right) & k \text{ is odd.} \end{cases}. \end{aligned} \tag{5.45}$$

As the eigenfunctions are orthogonal we note that we can write any $\mathbf{r}(s,t)$ as a weighted sum of eigenfunctions of the operator. Then the solution to the PDE is straightforward,

$$\mathbf{r}(s,t) = \sum_{k=1}^{\infty} a_k(0) \mathrm{e}^{\alpha\lambda_k t} \mathbf{v}_k(s). \tag{5.46}$$

If $\alpha > 0$ the curve will shrink, for $\alpha = 0$ it is constant and for $\alpha < 0$ it will expand.

Now let us also include the β-term, so only $\gamma = 0$. For the β- term we have a similar situation, as here we have the fourth derivative operator. The fourth derivative operator is nothing but the square of the second derivative operator, so the eigenvalues are squared and the eigenfunctions are the same. Here we get,

$$\mathbf{r}(s,t) = \sum_{k=1}^{\infty} a_k(0) \mathrm{e}^{(\alpha\lambda_k - \beta\lambda_k^2)t} \mathbf{v}_k(s). \tag{5.47}$$

Where if $\alpha - \beta\lambda_k > 0$ the curve will shrink, for $\alpha - \beta\lambda_k = 0$ it is constant and for $\alpha - \beta\lambda_k < 0$ it will expand.

We can also look at the case where $\beta = 0$, but $\gamma \neq 0$. This in particular illustrative as we compare it to a well known equation,

$$\frac{\partial \mathbf{r}(s,t)}{\partial t} = \alpha \frac{\partial^2 \mathbf{r}(s,t)}{\partial s^2} + \gamma \mathbf{F}(\mathbf{r}(s,t)), \text{ compared to } i\hbar \frac{\partial}{\partial t} \Psi(\mathbf{r},t) = \left[\frac{-\hbar^2}{2\mu} \nabla^2 + V(\mathbf{r},t) \right] \Psi(\mathbf{r},t). \tag{5.48}$$

Where the equation which we compare to is the time-dependent Schrödinger equation in position basis. Let us include the β-term, then if the force $\mathbf{F}$ is linearly dependent on $\mathbf{r}$, the solution can still be found rather easily,

$$\mathbf{r}(s,t) = \sum_{k=1}^{\infty} a_k(0) \mathrm{e}^{(\alpha\lambda_k - \beta\lambda_k^2 + \gamma)t} \mathbf{v}_k(s). \tag{5.49}$$

Where if $\alpha\lambda_k - \beta\lambda_k^2 + \gamma < 0$ the curve will shrink, for $\alpha\lambda_k - \beta\lambda_k^2 + \gamma = 0$ it is constant and for $\alpha\lambda_k - \beta\lambda_k^2 + \gamma > 0$ it will expand. If the force is some constant c we can write the solution, but it becomes hard to tell the exact behaviour of the curve.

$$\mathbf{r}(s,t) = \sum_{k=1}^{\infty} a_k(0) \mathrm{e}^{(\alpha\lambda_k - \beta\lambda_k^2)t} \mathbf{v}_k(s) + ct. \tag{5.50}$$

We can also condition the GVF algorithm to solve for a non-linear force. This is a difficult problem (as we know from quantum mechanics), so we will not solve that problem here. However it remains an interesting question.

All-in-all we have now gained some insight in the effect the choice of parameters can have. If we choose incorrectly the solution expands out of control, or the growth is prohibited yielding cells that are too small. Let us look at some of the results in Figures 5.2 and 5.3. Note that all the cells here are found quickly, and they also follow the cell shapes in the original image. Adding to that we have a handle to determine which cells are not correct, namely the ones with a too low isoperimetric quotient.

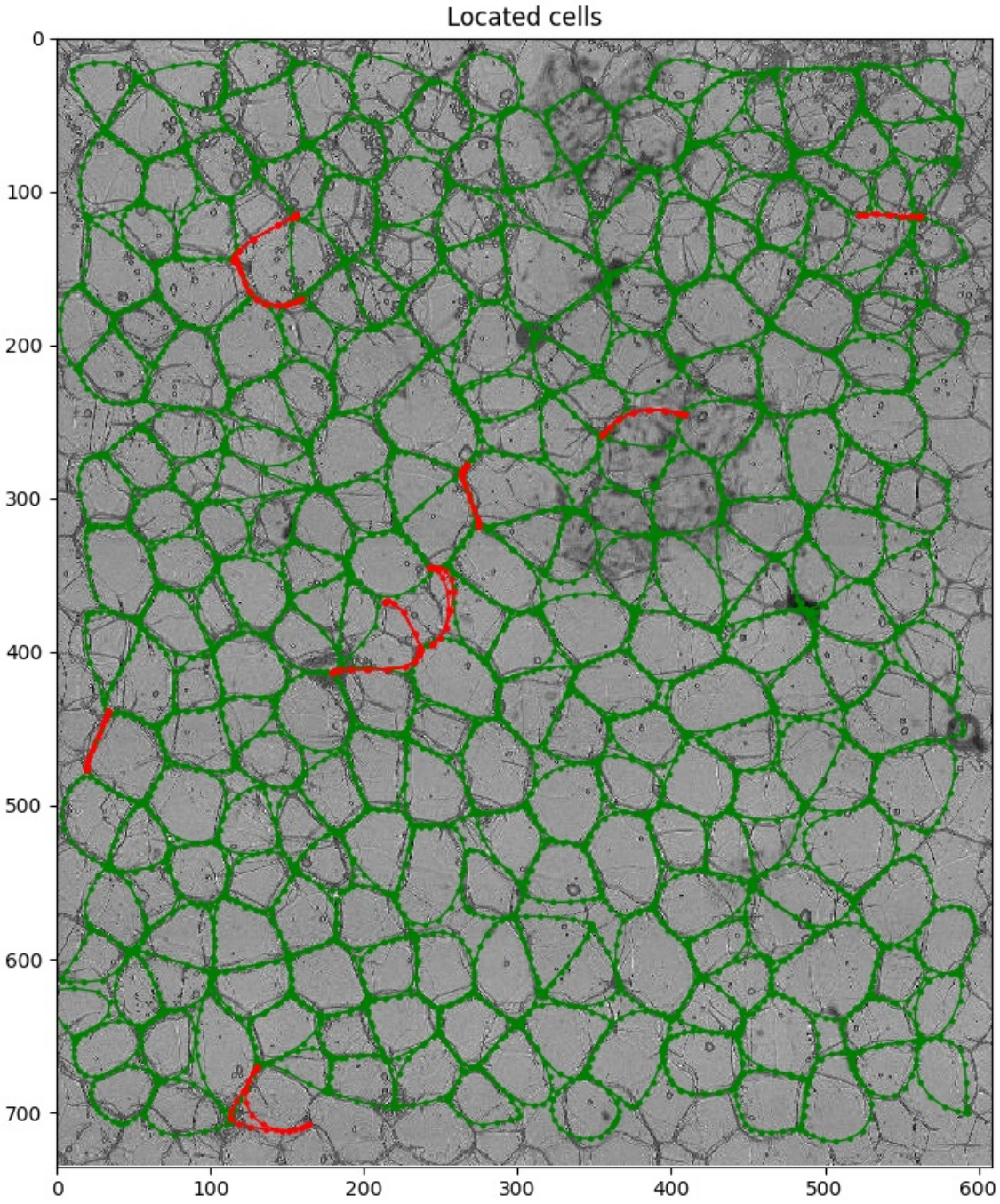
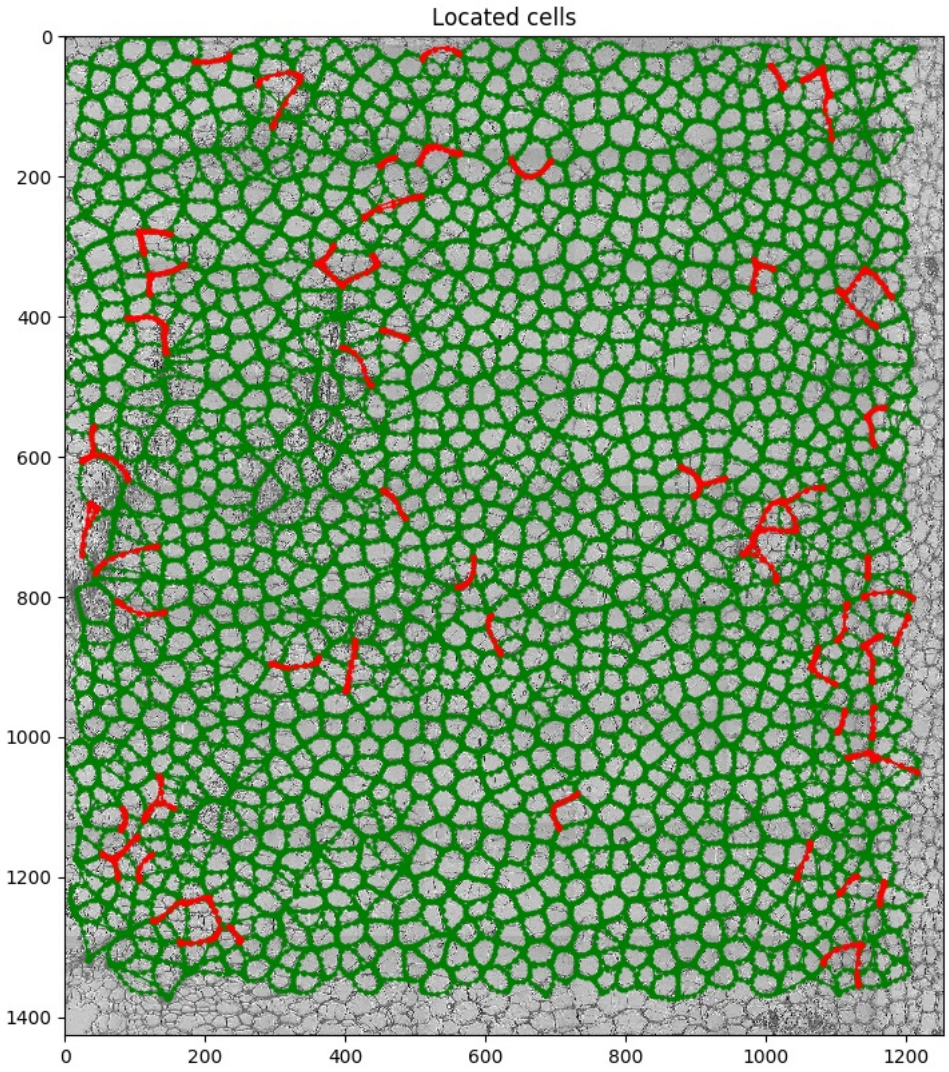
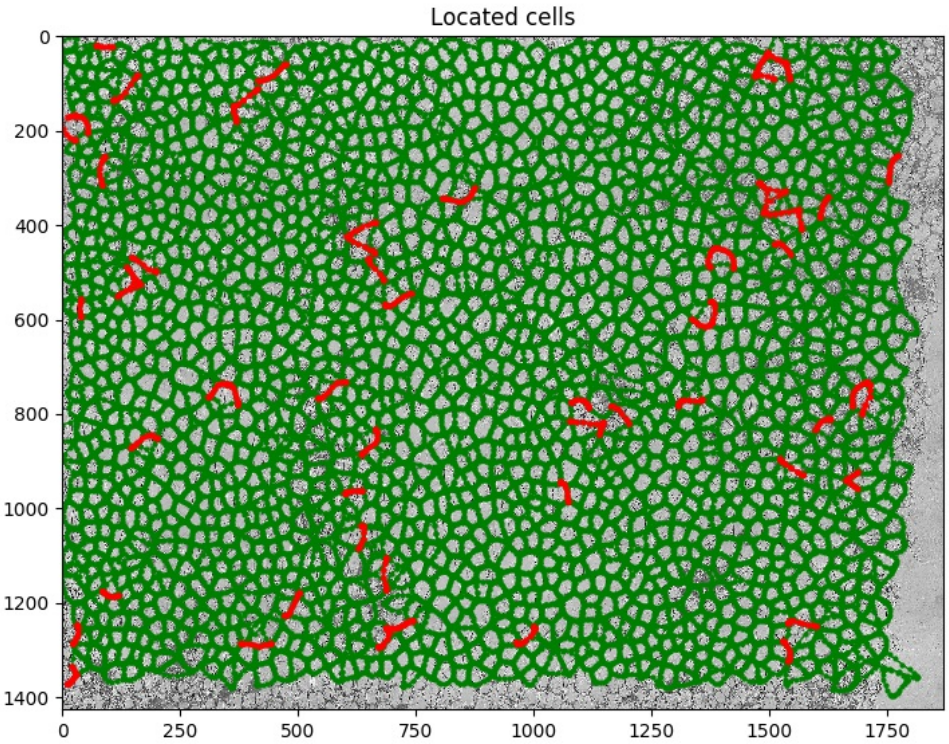


Figure 5.2: Here we see the resulting segmentation using the GVF algorithm. The red 'cells' are not satisfactory to our standard, that is their isoperimetric quotient is below 0.4. Which translates into the fact that they all are long, but have a very small surface area and can therefore not be regarded as cells. Note that this image is also cropped, some of the edge has been cut out by our cropping algorithm which is described in Appendix C. The original image can be seen in Appendix B, in Figure B.1.



(a) Now we implement the GVF method on a larger image and see that it still yields a good result. The original image is provided in the appendix as Figure B.4.



(b) We allow the GVF algorithm to work on this even larger image. The original image is given in the appendix as Figure B.5.

Figure 5.3: In these two example figures we have used the GVF method to find the cells. The right and bottom side are not covered up to the edge. This as we do not want to walk out of the array inside the program, therefore we stay clear from the sides.

5.5. Numerical solvers for the GVF equations

To find the vector flow field we need to solve the equations in Eq. (5.31). We will use a function from the scipy library, however there are many different solvers, we want to determine the fastest solver for our problem. We consider a selection of different solvers, note that the solution becomes harder to find when the value of μ increases. Our test criteria are, the image size and the value of μ. There are four suitable solvers on our list namely: GMRES, LGMRES, CG and BiCGstab.
For further details on each of these solvers we refer to [24]. We now test the solvers on some image and see which is the fastest.

μ	0.0	0.01	0.03	0.05	0.1	0.25	0.5
GMRES	0.60	0.86	1.09	1.20	1.41	2.18	3.19
LGMRES	0.33	0.63	0.64	0.74	0.85	1.02	1.22
CG	0.53	0.63	0.67	0.76	0.87	1.10	1.38
biCGstab	0.30	0.54	0.65	0.74	0.87	1.09	1.42

Table 5.1: In this table we see the processing time in seconds of each method for a given value of μ. The image on which we operate is 400×400 pixels in size.

μ	0.0	0.01	0.03	0.05	0.1	0.25	0.5
GMRES	3.15	4.58	5.81	6.55	8.11	12.64	19.72
LGMRES	2.00	3.53	3.60	4.74	4.73	7.24	9.15
CG	2.42	2.67	3.33	3.65	4.33	5.76	7.37
biCGstab	1.33	2.56	2.93	3.38	3.90	5.44	6.89

Table 5.2: In this table we have the same as in Table 5.1, but now for a larger image. The image on which we operate is 800×800 pixels in size.

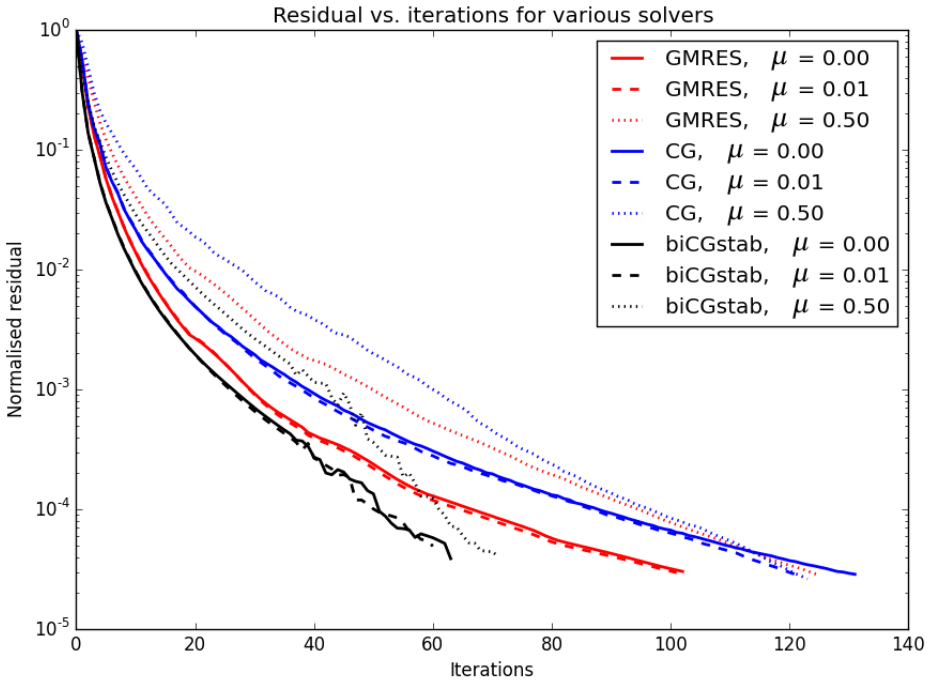


Figure 5.4: Here we see the residual vs. the number of iterations for three of the solvers operating on an 400×400 pixel image. Here we see that increased values of μ increase the number of iterations needed. Clearly the biCGstab solver takes the smallest number of iterations.

From Figure 5.4 and Tables 5.1 and 5.2 we learn that the fastest solver is the BiCGstab solver for both a small and large image. So we will employ it in all our further analysis.

6

Statistics of cell geometry

6.1. Introduction

For the cells that we locate using the GVF algorithm we want to calculate some geometric properties. To be precise we are interested in the cell area, perimeter and 'sphericity'. The population of cells provides us with a statistical sample that we can analyse. We will look into all three parameters one at a time. The three parameters are linked through the following definition. We note the unit of perimeter is pixels, and for the area it is pixels squared.

Definition 6.1.1. The **isoperimetric quotient** φ (sometimes referred to as 'sphericity', but that only works in 3D) or rather its square root, is calculated in 2D by the following formula:

$$\varphi = \frac{2\sqrt{\pi A}}{L}. \tag{6.1}$$

Here A is the area of the given cell, and L the perimeter. So this is the ratio between the perimeter of a circle with the same area as the particle and the perimeter of the actual particle. In 3D the formula would read:

$$\psi = \frac{\pi^{\frac{1}{3}}(6V)^{\frac{2}{3}}}{A}. \tag{6.2}$$

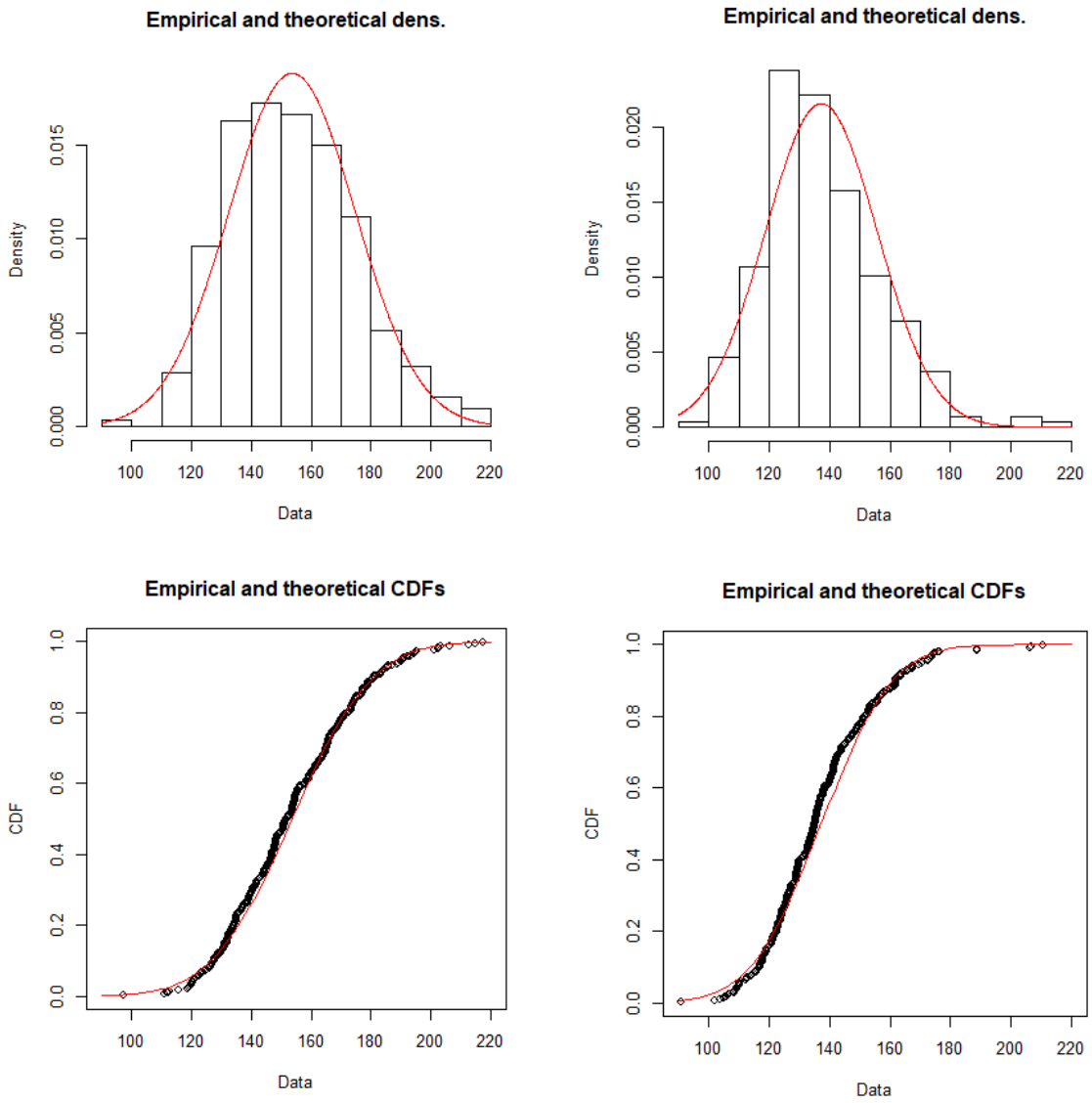
The 3D formula is the actual formula for the sphericity of an object.

6.2. Cell perimeter

We begin with the perimeter, because in the first place it can be determined quickly. Secondly the perimeters seem to be distributed with the normal distribution, as this is a well known distribution it provides us a good starting point for further analysis. Note that there is one big difference, the perimeter can never be negative, whereas the normal distribution would allow for negative values.
When we look at the results in Figure 6.1 the perimeter seems to be normally distributed. We compare the fraction of counts in the usual standard deviation brackets for the data from the potato pith cells:

Standard deviation	Theoretical fraction	Observed fraction
$\pm\,\sigma$	0.6827	0.6886
$\pm\,2\sigma$	0.9545	0.9581
$\pm\,3\sigma$	0.9973	0.9934

Now to prove that the perimeter is distributed with a normal distribution we can for instance use the χ^2- test (from the scipy stats package). When we do this, in particular for the large dataset the result is that the perimeter is not likely to be normally distributed. So again we point out that the cell perimeter can never be negative valued, therefore the data is not fully symmetrical as it would be in the normal distribution. But given that the difference is very small, for simplicity we will assume that the perimeter is normally distributed.



(a) A histogram and cumulative density function of potato cortex cells. The original image is provided in Figure B.1.

(b) A histogram and cumulative density function of potato pith cells. The original image is given in Figure B.6.

Figure 6.1: Here we see the distribution of cell perimeters for two different images. This shape is a general feature for all cell perimeters.

6.3. Cell area

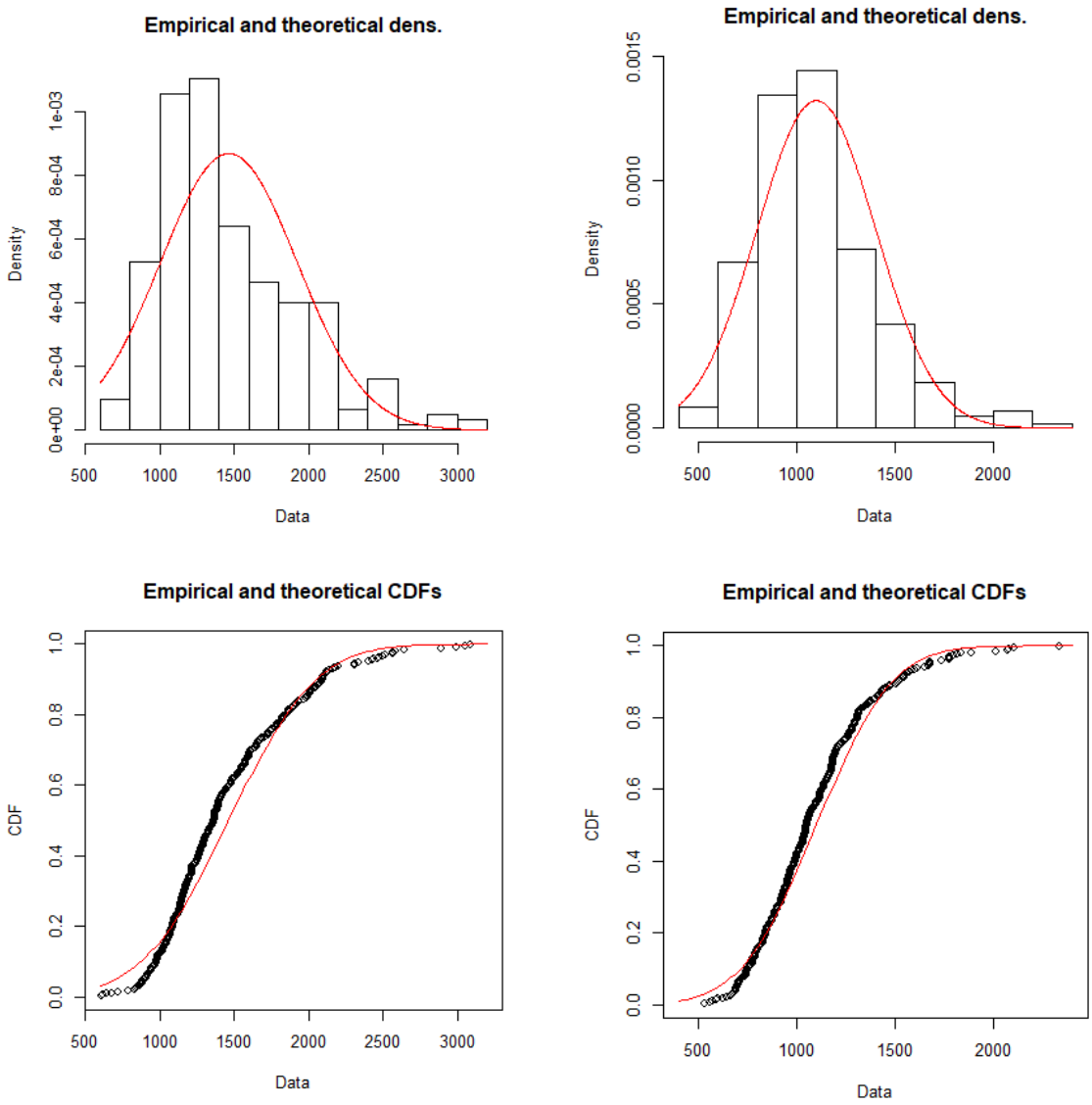
In a similar way we can look at the distribution of the cell area, a quick glance (see Figure 6.2) already suggests that the distribution is not the normal distribution. So we will build a function of a random variable, as we know that the perimeters are normally distributed. Area and perimeter are linked via the isoperimetric quotient.

6.3.1. Calculating the area

In order to quickly calculate the cell area we use the shoelace formula, which is a special case of Green's formula. This formula can be implemented as our cells are simple polygons. For each cell we have a number of vertices i which we label $1,\dots,n$. Then the surface area is given by,

$$A = \frac{1}{2}\left|\sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n\right|. \tag{6.3}$$

Note that the coordinates need to be ordered clockwise.



(a) A histogram and cumulative density function of potato cortex cells. The original image is provided in Figure B.1.

(b) A histogram and cumulative density function of potato pith cells. The original image is given in Figure B.6.

Figure 6.2: When we look at the histogram and cumulative density function we note that the cell area is probably not distributed with a normal distribution. However as an initial guess it would not be the very worst approximation to the real distribution function.

6.3.2. As a function of the perimeter

Given the statistical results of the analysis we can reasonably assume that the perimeter of the cells is normally distributed. However the area and especially the isoperimetric quotient are not. As these three cell parameters are all random variables we can attempt to build a composite function.

As we have stated the area of the cells does not seem to be distributed with a normal distribution. Therefore we will build a new probability density function (PDF), therefore we assume that the circumferences are indeed distributed normally. For simplicity we will also assume that the isoperimetric quotient is constant. So we have $A = \chi L^2$, where $\chi = \varphi^2/(4\pi)$ and $L \sim \mathcal{N}(\mu_l, \sigma_l)$. Where φ is the isoperimetric quotient redefined for 2D as in Definition 6.1.1.

Now to construct a new PDF we need to calculate the cumulative density function (CDF) or simply integrate the pdf of the circumference. Also read *Mathematical statistics and data analysis* [25, pp. 58-63]. First

we note that $\frac{4\pi A}{\psi} = L^2$ here $\psi = \varphi^2$. We begin with the CDF of the area distribution:

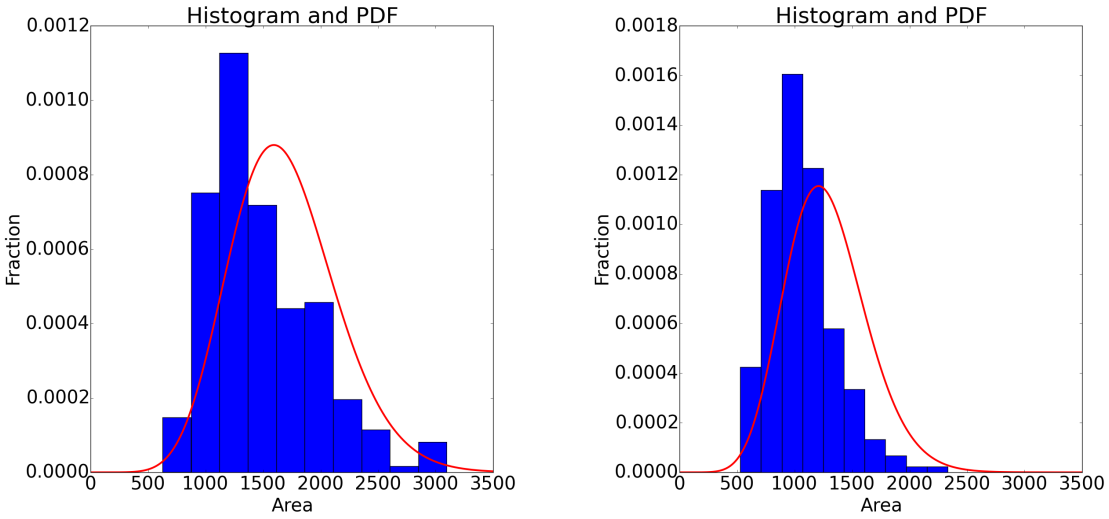
$$\begin{aligned}
F_A(a) &= P(A \le a) \\
&= P\left(-\sqrt{\frac{4\pi a}{\psi}} \le L \le \sqrt{\frac{4\pi a}{\psi}}\right) \\
&= P\left(\sqrt{\frac{4\pi a}{\psi}} \le L\right) - P\left(-\sqrt{\frac{4\pi a}{\psi}} \le L\right) \\
&= \int_{-\infty}^{\sqrt{\frac{4\pi a}{\psi}}} \frac{1}{\sigma_l\sqrt{2\pi}} e^{-\frac{(l-\mu_l)^2}{2\sigma_l^2}}\,\mathrm{d}l - \int_{-\infty}^{-\sqrt{\frac{4\pi a}{\psi}}} \frac{1}{\sigma_l\sqrt{2\pi}} e^{-\frac{(l-\mu_l)^2}{2\sigma_l^2}}\,\mathrm{d}l \\
&= \frac{1}{2}\left\{\left[\operatorname{erf}\left(\frac{\mu_l + \sqrt{\frac{4\pi a}{\psi}}}{\sqrt{2}\sigma_l}\right) - 1\right] - \left[\operatorname{erf}\left(\frac{\mu_l - \sqrt{\frac{4\pi a}{\psi}}}{\sqrt{2}\sigma_l}\right) - 1\right]\right\} \\
&= \frac{1}{2}\left\{\operatorname{erf}\left(\frac{\mu_l + \sqrt{\frac{4\pi a}{\psi}}}{\sqrt{2}\sigma_l}\right) - \operatorname{erf}\left(\frac{\mu_l - \sqrt{\frac{4\pi a}{\psi}}}{\sqrt{2}\sigma_l}\right)\right\}.
\end{aligned} \tag{6.4}$$

Then we need to take the derivative with respect to a:

$$\frac{\mathrm{d}}{\mathrm{d}a} F_A(a) = f_A(a) = \frac{1}{2}\left[\frac{\sqrt{2}}{\psi\sigma_l\sqrt{\frac{a}{\psi}}}\left(e^{-\frac{\left(\sqrt{\frac{4\pi a}{\psi}} + \mu_l\right)^2}{2\sigma_l^2}} + e^{-\frac{\left(\sqrt{\frac{4\pi a}{\psi}} - \mu_l\right)^2}{2\sigma_l^2}}\right)\right], \quad a \ge 0. \tag{6.5}$$

Which is the new PDF for the cell area.

In this PDF we see that entering the value $a = 0$ gives a singularity. Given the parameters in our situation it is not relevant for the dataset. This is quite possibly an artefact of the perimeter not being completely normally distributed.



(a) A histogram and our new PDF for the cell area of potato cortex cells. The original image is provided in Figure B.1.

(b) A histogram and PDF for the cell area of potato pith cells. The original image is given in Figure B.6.

Figure 6.3: Here we see the distribution of cell area for two different images. Note that the shape of the new PDF follows the histogram much better than the normal distribution, however it is still not perfect.

When we compare this distribution to the actual data set in Figure 6.3 we see that this distribution function's shape quite nicely follows the data, however it does seem to be a little too much to the right.

6.4. isoperimetric quotient

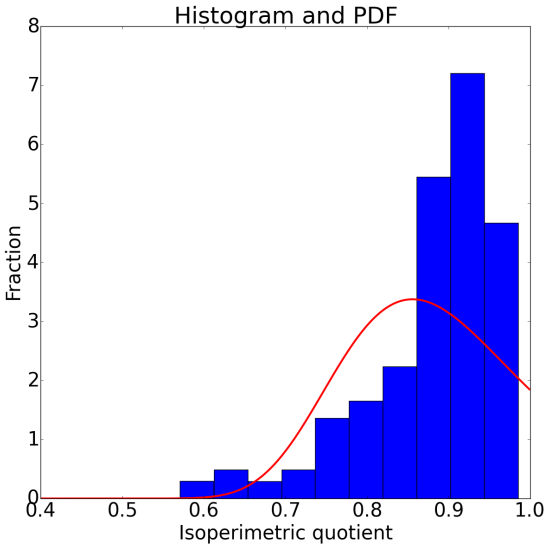
The distribution of the isoperimetric quotient is quite hard to determine. We first try to build a function of a random variable, just like for the cell area. From Equation (6.1) we note that $\varphi = \frac{2\sqrt{\pi A}}{L}$, with $L \sim \mathcal{N}(\mu_l, \sigma_l)$. Let us first assume that A is a constant. Also take note that $|\varphi| \leq 1$, which is something we do not demand in the resulting equation.
For notational purposes we define $\Omega = 2\sqrt{\pi A}$.

$$\begin{aligned} F_\Phi(\varphi) &= P(\Phi \leq \varphi) \\ &= P\left(\frac{\Omega}{\varphi} \geq L\right) \\ &= \int_{\frac{\Omega}{\varphi}}^{\infty} \frac{1}{\sigma_l\sqrt{2\pi}} e^{-\frac{(l-\mu_l)^2}{2\sigma_l^2}} \, \mathrm{d}l \\ &= -\frac{1}{2}\left[\operatorname{erf}\left(\frac{\Omega - \varphi\mu_l}{\sqrt{2}\varphi\sigma_l}\right) - 1\right]. \end{aligned} \tag{6.6}$$

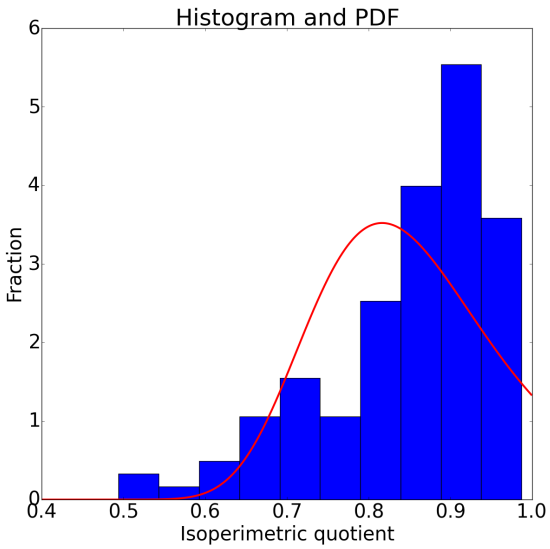
Which leads us to conclude that the new PDF is,

$$\frac{\mathrm{d}}{\mathrm{d}\varphi} F_\Phi(\varphi) = f_\Phi(\varphi) = \frac{\Omega e^{-\frac{(\Omega-\varphi\mu_l)^2}{2\sigma_l^2\varphi^2}}}{\varphi^2\sqrt{2\pi}\sigma_l}, \ \varphi \geq 0. \tag{6.7}$$

This resulting PDF does not accurately describe the dataset at all, however its shape is promising. We must not forget that the cell perimeter distribution is not entirely normal, neither is the cell area remotely constant. A more detailed calculation including the stochastic nature of the cell area could improve the result greatly.



(a) A histogram and our new PDF for the isoperimetric quotient of potato cortex cells. The original image is provided in Figure B.1.



(b) A histogram and PDF for the isoperimetric quotient of potato pith cells. The original image is given in Figure B.6.

Figure 6.4: Here we see the distribution of cell area for two different images. Note that the shape of the new PDF follows the histogram much better than the normal distribution, however it is still not perfect.

6.5. Scatter plot

Here we also include some scatter plots, to further illustrate the data. Note that all the data points are located on a parabolic surface that is described be Equation (6.1) from the definition of the isoperimetric quotient.

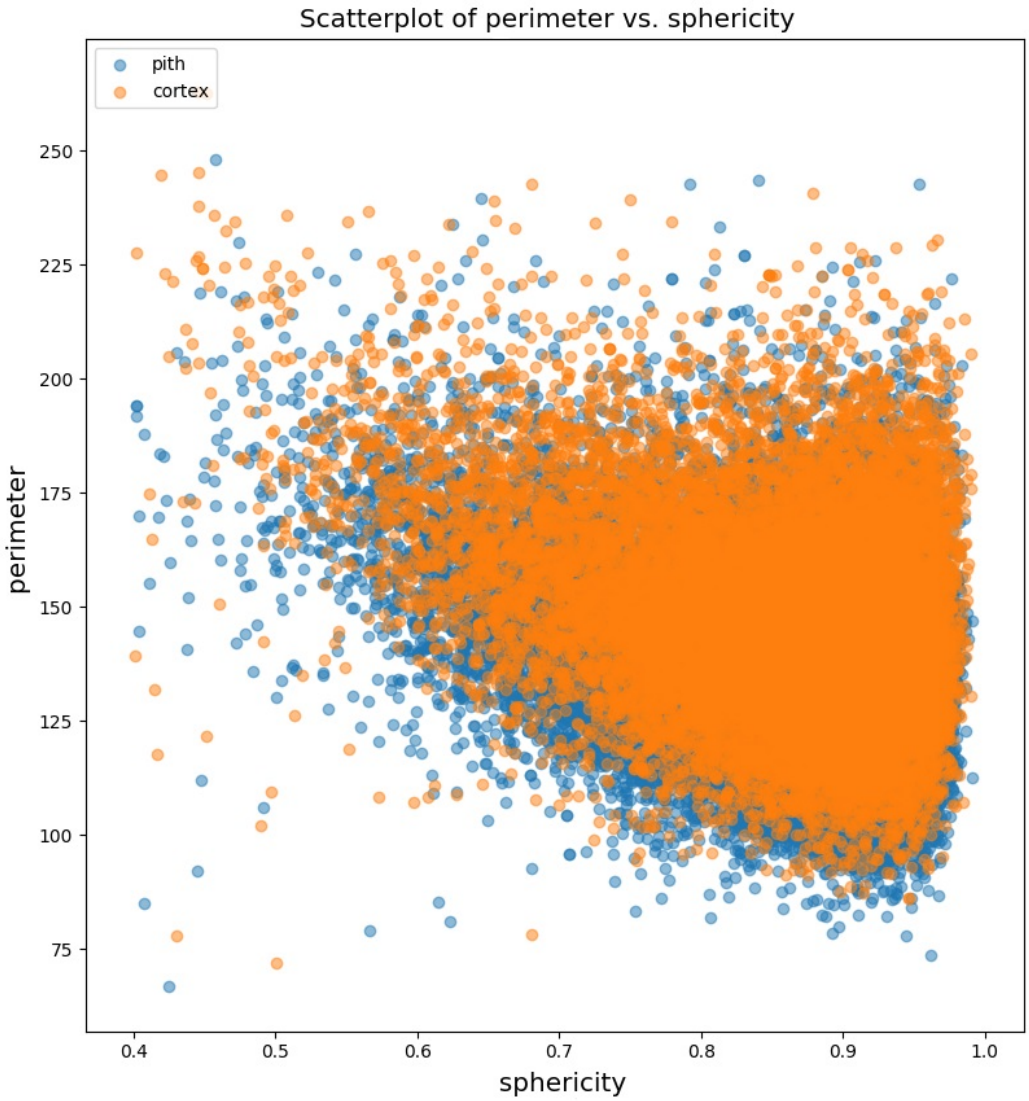


Figure 6.5: Here we see the scatter plots of the cell perimeter vs. isoperimetric quotient. The orange data belongs to cortex cells and the blue to pith cells. We note two things; Firstly, both data sets have the same general shape. Secondly, cortex cells are (on average) bigger. The data was taken and combined from 20 different images.

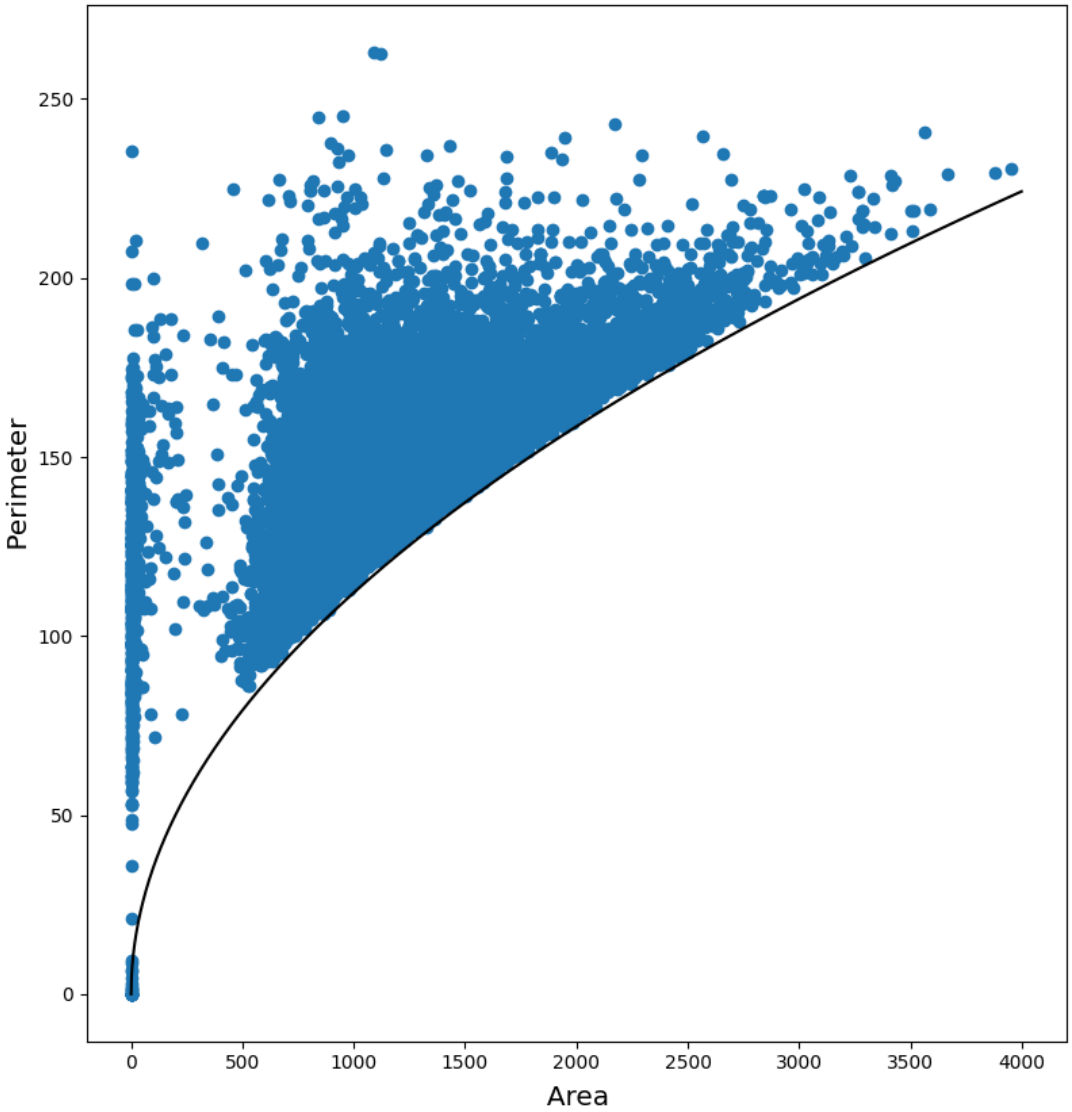


Figure 6.6: Scatter plot for the cortex cell, for area vs. perimeter. We point out that the black line is Eq. (6.1) with $\varphi = 1$. Note that there are two clusters, one for cells with a small area. These cells are the red long stretched cells that we saw in for instance Figure 5.2. The other cluster contains the correct cells.

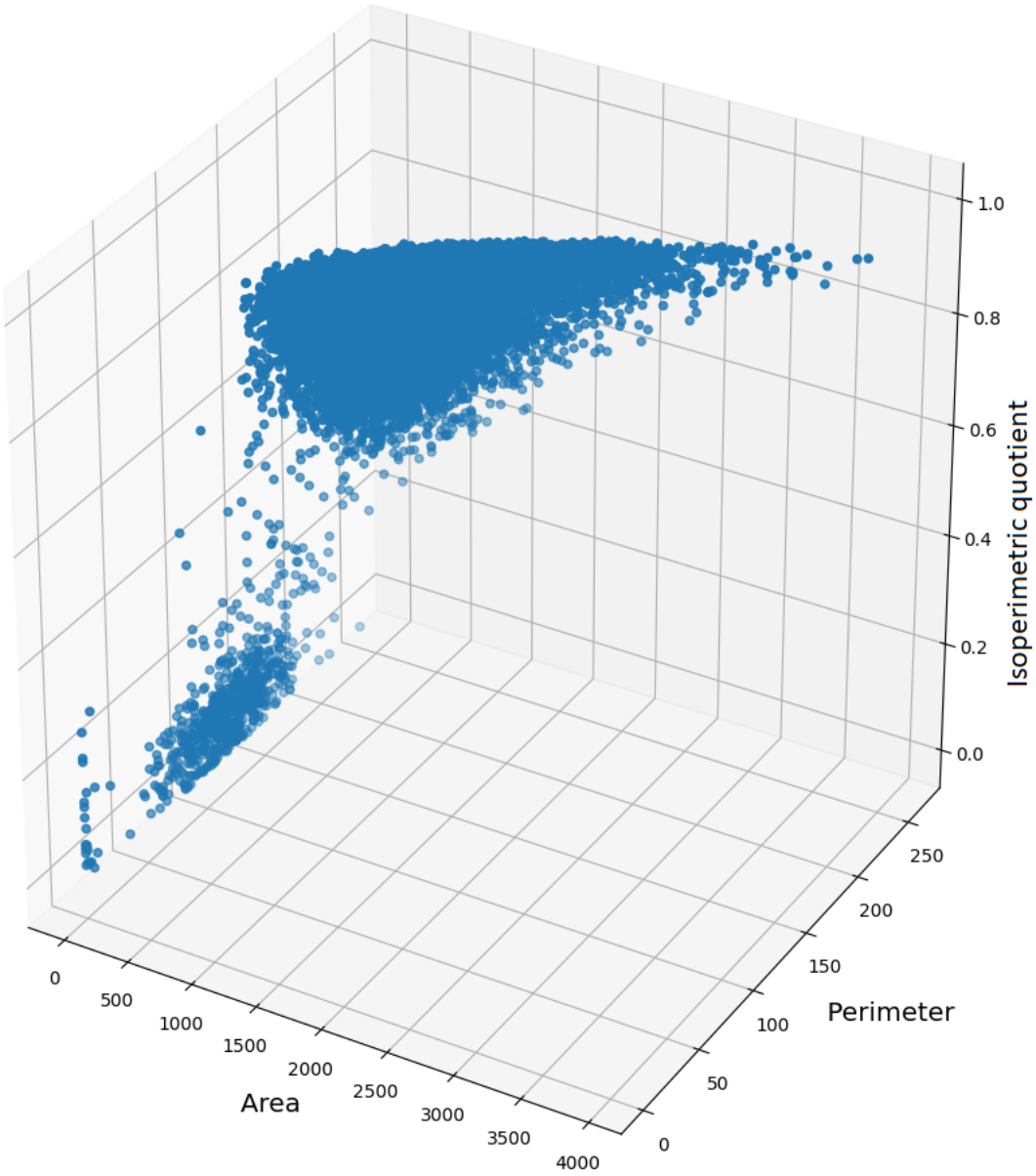


Figure 6.7: Again we see the data from the cortex cells. Now notice how the cells with low isoperimetric quotient are clustered together. Also note that the other cells are clustered together as well.

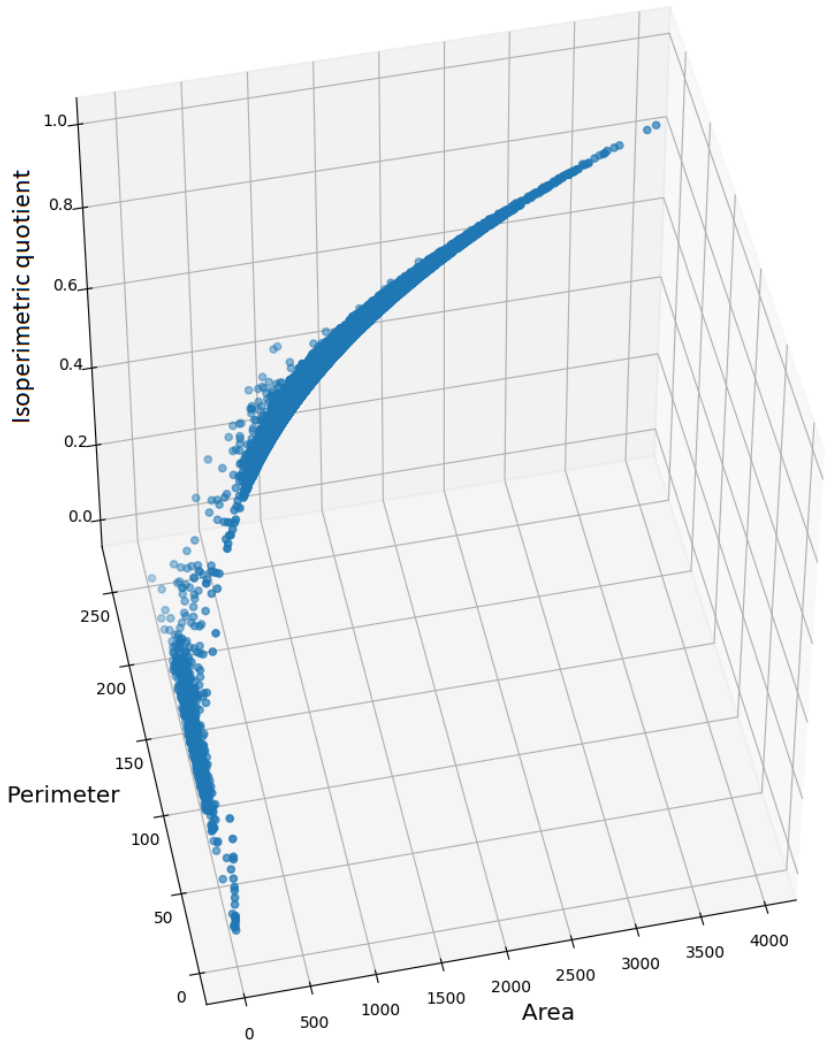


Figure 6.8: Here we look at how the data points follow the parabolic surface, again these are cortex cells. The surface area is described by Equation 6.1, and on this surface we have the two clusters of 'good' and 'bad' cells.

7

Conclusion & Recommendations

7.1. Conclusion

To conclude we have looked thoroughly at the problem of "*Automated analysis of microscopic images of cellular tissues*". During the research we have learned a lot and came up with a method to find cells. Let us briefly retrace our paths.

Firstly we have sketched our problem and saw what challenging task lay ahead of us. How do we take care of the impurities in the image such that we find as many real cells as possible, and still end up with a practical algorithm.

Using simple techniques and intuition we set out to segment the image. It is clear that the watershed method on the raw image data yields poor and impractical results. Following this lead it is also clear that preconditioning the image does not necessarily give a reliable or workable starting point. Based on finding cell centres and vertices in the image we learn that the fitting of polygons or ellipses also do not provide useful results.

However, the cell centres and vertices are very good anchor points in the image. So a further more in-depth review was done in Chapter 3, there the Weierstraß transformation has been studied. Here we have learned some details on the localisation of the centres and vertices. Based on these points that we have named 'points of interest', or POI's for short, it is possible to reconstruct cells.

In Chapter 4 several methods using the POI's are reviewed. It is for instance possible to connect the vertices. In other methods the POI's can be made into attracting and repelling objects in the image, that act on certain freely moving particles. When these free particles are linked together as a snake, as described in Chapter 5, we have yielded good and practical results.

From the resulting segmentations we have extracted sets of statistical data to learn about the cell perimeter, cell area and isoperimetric quotient. We see that the cell perimeter is more or less normally distributed, using that assumption we have been able to construct some simple probability density functions for the cell area and isoperimetric quotient.

7.2. Recommendations

We can locate the cell in the image, however there is still room for improvement. For instance the GVF method as we have used it can be used on differently preprocessed images as well. It is very interesting to see how different preprocessing methods effect the outcome.

A more pressing problem is that some cells are located twice, as they contain two bright spots. The result is that two smaller cells are found, when this happens to often it has noticeable consequences for the statistics. Something we want to prevent at all costs. In a similar fashion certain small cell might be overlooked. One approach could be fine tuning the cell centre finding algorithm.
Another practical issue is that the images contain areas in which there are a lot of impurities, or where there

is clearly no potato tissue to be found. To lower the number of false positives we would like to find improved cropping techniques.

The constructed probability density functions for the cell area and isoperimetric quotient were made in a simple manner. They make use of the graphically reasonable assumption that the cell perimeter is normally distributed, however the χ^2- test shows that is not entirely correct. This shows in particular for the PDF of the isoperimetric quotient, in a further study it would be recommended to focus on deriving better PDF's.
A large leap forward would be to at least include the full behaviour of the cell area, rather than assuming it to be constant.

A

Theory

A.1. Introduction

In this chapter we will introduce basic theory, i.e. the building blocks and background to the techniques that we will use to analyse the images. For instance we introduce the used image filters as they are well known tools for image enhancement. The actual implementation and more specific theory will be given in later chapters when it is necessary.

A.2. Digital image filters

In the process of analysing the various images we can choose from a broad array of filters and morphologies. Before we do that it is important to note we essentially work with the grey scale of the images. In the grey scale the colour ranges from black to white, which translates into a brightness value between 0 and 1 respectively. For a more detailed and concise overview of filters and morphologies one can read *Digital Image Processing* [12].

A.2.1. Convolution

The most important operation that we use when implementing any filter is the convolution operation. We will use the standard notation and definition, which in for discrete signal in 1D reads:

$$F[n] = x[n] * f[n] = \sum_{k=-\infty}^{\infty} x[k] f[n-k], \tag{A.1}$$

where we say that $F[n]$ is the convolution between $x[n]$ and $f[n]$.

Since we will work with images, we will use a 2D convolution which is then given as:

$$F[n_1, n_2] = x[n_1, n_2] * * f[n_1, n_2] = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x[k_1, k_2] f[n_1 - k_1, n_2 - k_2]. \tag{A.2}$$

When we want to compute the convolution analytically, we need to use the discrete Fourier transform on the circular convolution. The circular convolution is nothing but a convolution with a periodic signal:

$$F[n_1, n_2] = x[n_1, n_2] * * f[n_1, n_2] = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \left(x[k_1, k_2] \sum_{m_1=-\infty}^{\infty} \sum_{m_2=-\infty}^{\infty} f[n_1 - k_1 - m_1 N, n_2 - k_2 - m_2 N] \right), \tag{A.3}$$

where N is the period of the periodic signal.

The discrete Fourier transform reduces the circular convolution to, what we might refer to as; 'pixel' space, to a product in frequency space.

$$F[n_1, n_2] = x[n_1, n_2] * * f[n_1, n_2] \Leftrightarrow F(\omega_1, \omega_2) = x(\omega_1, \omega_2) f(\omega_1, \omega_2). \tag{A.4}$$

In essence we can see that the convolution is nothing more than placing a certain weighting mask on the original image. For that mask there is some centre coordinate, on this coordinate we get the weighted sum of all the pixels covered by the mask. Let us examine a simple example in Figure A.1.

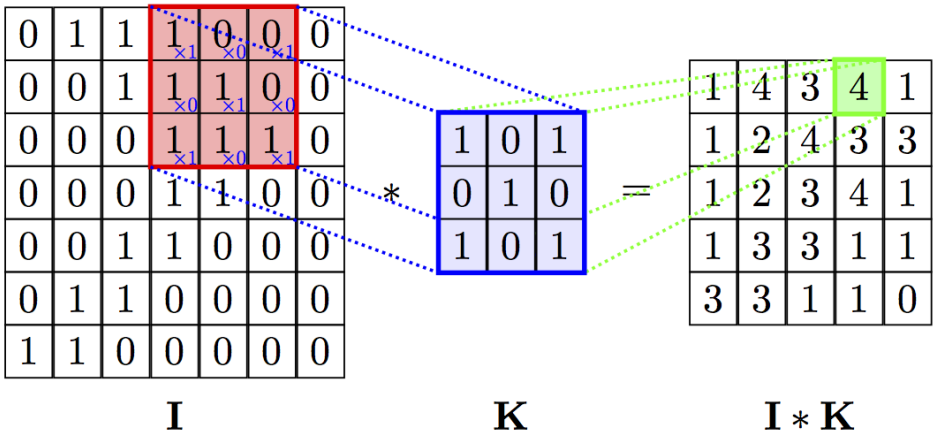


Figure A.1: A simple illustration of how the convolution between two matrices works. Source: *Deep learning for complete beginners: Using convolutional nets to recognise images*, Cambridge Coding Academy [13].

When doing this there are still various approaches to dealing with the image boundary. In the example of Figure A.1 the boundary elements are not considered at all, but we want to retain the original image size. So we must extend the original image matrix, in order to supply the convolution with the necessary data points. We make a short list of options (cf. [12, pp. 114, fig. 5.22]):

1. Constant value: Around the original image we add a region containing some fixed value.

2. Extend with the nearest value: Beyond the borders of the image we add new pixels with the value of the nearest border pixel.

3. Mirror the image: Here we simply mirror the image matrix into the border areas.

4. Reflect the image: This might seem to be exactly the same as mirroring. Here we also copy the boundary pixels, whereas we do not in mirroring.

5. Duplicate the image: Here we paste the image on its own borders.

NB. The default setting in SciPy is 'reflect'.

For a more detailed explanation of the convolution operation (in 1D in particular) and its applications in signal processing see *Signals and systems* [14].

A.2.2. Local peaks and the binary maximum filter

In order to find the cell centres, vertices or boundaries we need to extract local maxima from the transformed image I. For that purpose we write a short algorithm, which makes use of a binary maximum filter and binary erosion in order to find local extrema. First we define the maximum filter:

Definition A.2.1. The **maximum filter** is a nonlinear image filter, in which the output array carries the maximum of the masked neighbourhood of each element. So for a mask M we have

$$I'(u,v) = \max_{(i,j)\in M} \{I(u+i, v+j)\}. \tag{A.5}$$

Where $(i, j) = (0,0)$ is the central element of the mask M. (Further details cf. [12, pp. 105])

Then we define the erosion morphology

Definition A.2.2. The **erosion morphology filter** only preserves an element if and only if the mask E is contained at its given position. The filter then is defined as

$$I \ominus E \equiv \left\{ \mathbf{s} \in \mathbb{Z}^2 | E_{\mathbf{s}} \subseteq I \right\}. \tag{A.6}$$

(cf. [12, pp. 186])

Using these two operations we can find local maxima in the 2D image if we use a 3×3 mask containing ones in the maximum filter. Once this has been done we can compare the filtered image with the original. The elements that are strictly equal are maxima. In order to make sure we do not retain artefacts from the maximum filter we need to delete background features using the erosion filter. A short Python code for implementing this can be found in Listing A.1. Now this code finds local maxima, but one can straightforwardly see that local minima are maxima in the inverted image, i.e. $1 - I$.

```
from scipy import ndimage

def detect_peaks(image):
    neighborhood = ndimage.morphology.generate_binary_structure(2,2)
    local_max = ndimage.filters.maximum_filter(image, footprint=neighborhood)==image

    background = (image==0)
    eroded_background = ndimage.morphology.binary_erosion(background, structure=neighborhood, border_value
     =1)

    detected_peaks = local_max ^ eroded_background

    return detected_peaks
```

Listing A.1: Local peak finding function

When we use a different mask, in which all the non-zero elements of the mask are on one line (in particular a line through the centre) we can find minima in a direction. Which on the 3×3 mask means: row extrema, column extrema and extrema in both diagonal directions.

When we try to find the local extrema on the Weierstraß transformed image we can find approximated cell centres and vertices, as well as cell boundaries.

A.3. Minimising a functional

In the active contour algorithms we try to solve an energy functional. Which in principle is also what we do when we are looking at the electrodynamic model. Both given in Chapter 4. For the gradient vector field we need to do this globally, this requires us to use a formal method which we want to bring into mind again. For a more detailed study we refer to *Numerical methods in scientific computing* [15], in particular chapter 5 of the book is important here.

Let us now work out the standard method for a general functional. Given a functional $J(u)$, where we minimise over u.

$$J(u) = \int_\Omega \left[Lu + uf + cu \right] \mathrm{d}\Omega. \tag{A.7}$$

Where L is some operator (e.g. n-th order derivative) working on u, f is a function on the domain Ω and c is a scalar.

The idea is that we assume u to be the function that minimises J, then we introduce a small perturbation ϵv. We derive with respect to ϵ take $\epsilon = 0$ and set the expression equal to 0.

$$\frac{\mathrm{d}}{\mathrm{d}\epsilon} J(u + \epsilon v)\bigg|_{\epsilon=0} = 0. \tag{A.8}$$

Here we do a step-by-step analysis, so first we introduce the perturbation.

$$J(u + \epsilon v) = \int_\Omega \left[L(u + \epsilon v) + (u + \epsilon v) f + c(u + \epsilon v) \right] \mathrm{d}\Omega. \tag{A.9}$$

Now let L for instance be the square of the first derivative (nabla operator).

$$\begin{aligned} J(u+\epsilon v) &= \int_\Omega \left[|\nabla(u+\epsilon v)|^2 + (u+\epsilon v)f + c(u+\epsilon v)\right] \mathrm{d}\Omega \\ &= \int_\Omega \left[|\nabla u|^2 + 2\epsilon \nabla u \cdot \nabla v + \epsilon^2 |\nabla v|^2 + (u+\epsilon v)f + c(u+\epsilon v)\right] \mathrm{d}\Omega. \end{aligned} \tag{A.10}$$

Derivative with respect to ϵ:

$$\frac{\mathrm{d}}{\mathrm{d}\epsilon} J(u+\epsilon v) = \int_\Omega \left[2\nabla u \cdot \nabla v + 2\epsilon |\nabla v|^2 + vf + cv\right] \mathrm{d}\Omega. \tag{A.11}$$

Then we set $\epsilon = 0$:

$$\left.\frac{\mathrm{d}}{\mathrm{d}\epsilon} J(u+\epsilon v)\right|_{\epsilon=0} = \int_\Omega \left[2\nabla u \cdot \nabla v + vf + cv\right] \mathrm{d}\Omega. \tag{A.12}$$

Now we need to use partial integration in 2D, which is a result of Green's theorem [16, pp. 147-158]:

$$\left.\frac{\mathrm{d}}{\mathrm{d}\epsilon} J(u+\epsilon v)\right|_{\epsilon=0} = \int_\Omega \left[-2v\Delta u + vf + cv\right] \mathrm{d}\Omega + \int_\Gamma v \nabla u \cdot \mathbf{n} \mathrm{d}\Gamma = 0, \forall \mathrm{v}, \tag{A.13}$$

where $\Delta = \nabla^2$.
Then according to Dubois-Reymond's lemma we get a PDE and boundary conditions:

$$\begin{aligned} 2\Delta u - c &= f, \\ \nabla u \cdot \mathbf{n}\Big|_\Gamma &= 0. \end{aligned} \tag{A.14}$$

Alternatively we can also use the Dirichlet boundary condition,

$$u\Big|_\Gamma = 0. \tag{A.15}$$

.

B

Some of the analysed images

Here we want to display some of the original images that we have worked with, this to bestow upon the reader the means to check out the results in more detail.

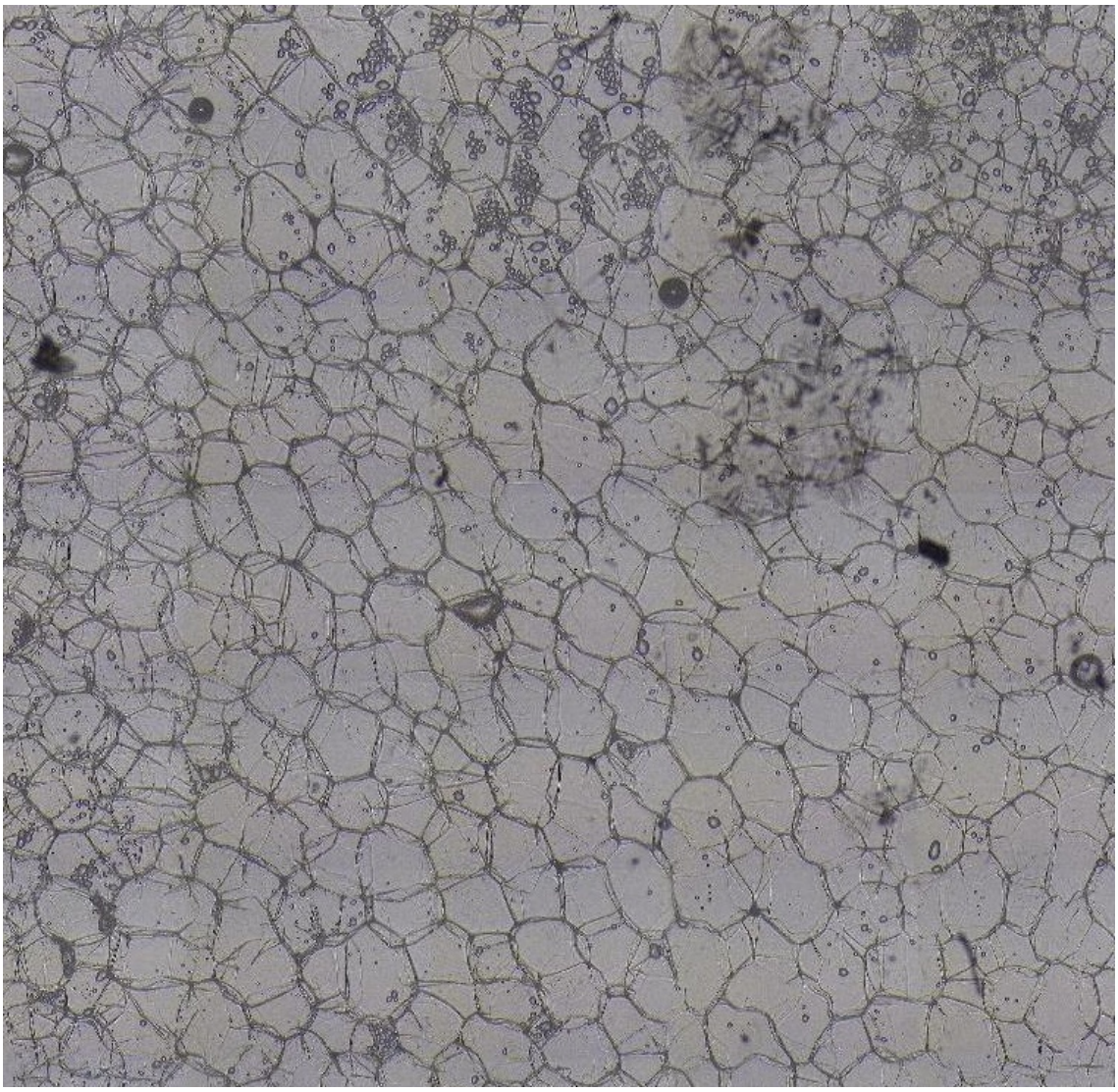


Figure B.1: Here we see an image originating from some potato cortex. We have used it, or part of it, in many instances. Named: cortex2.cropped.jpg

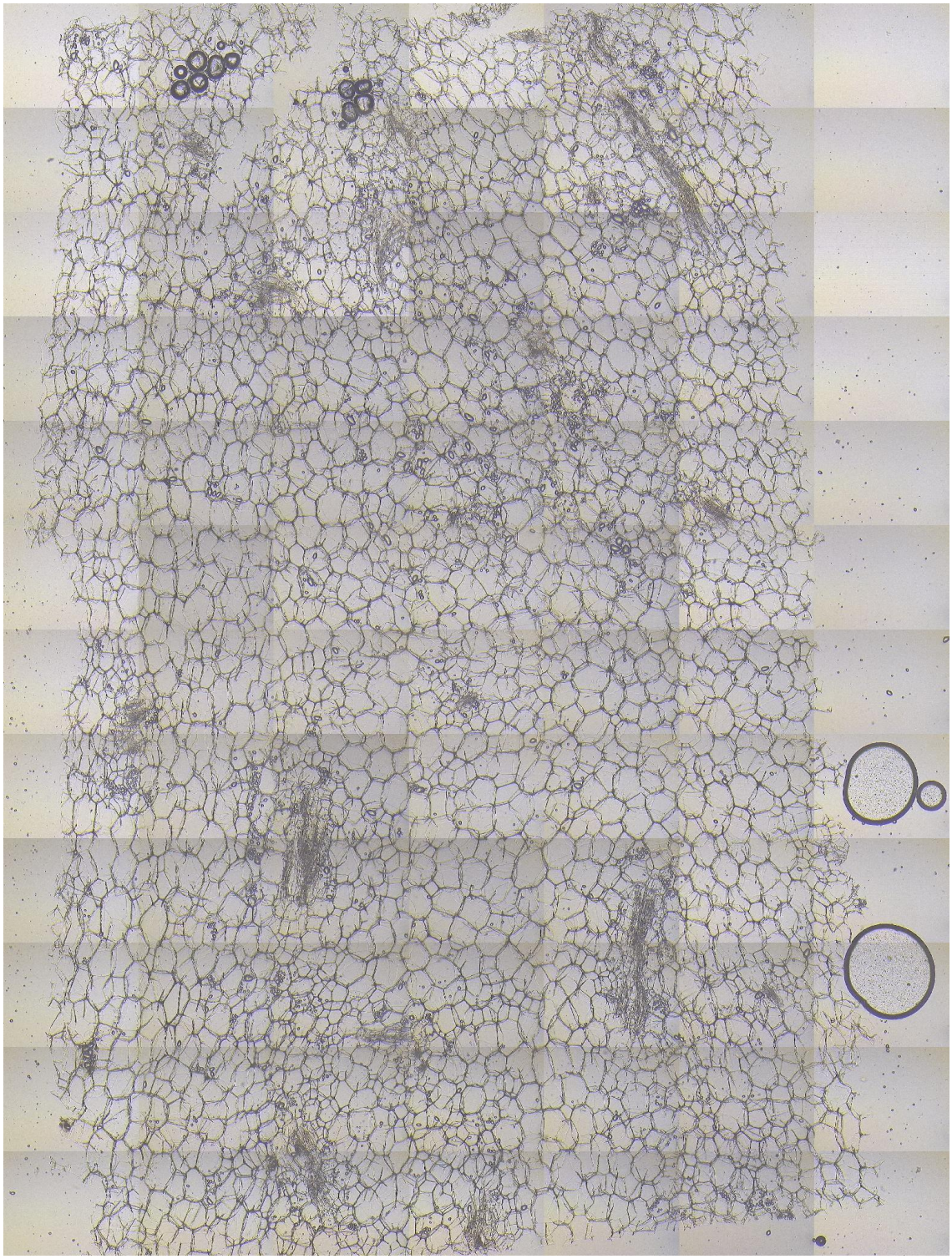


Figure B.2: Original image of figure 5.1b. Named: 101;US;2017;field11;T11;Pith.rep1.jpg

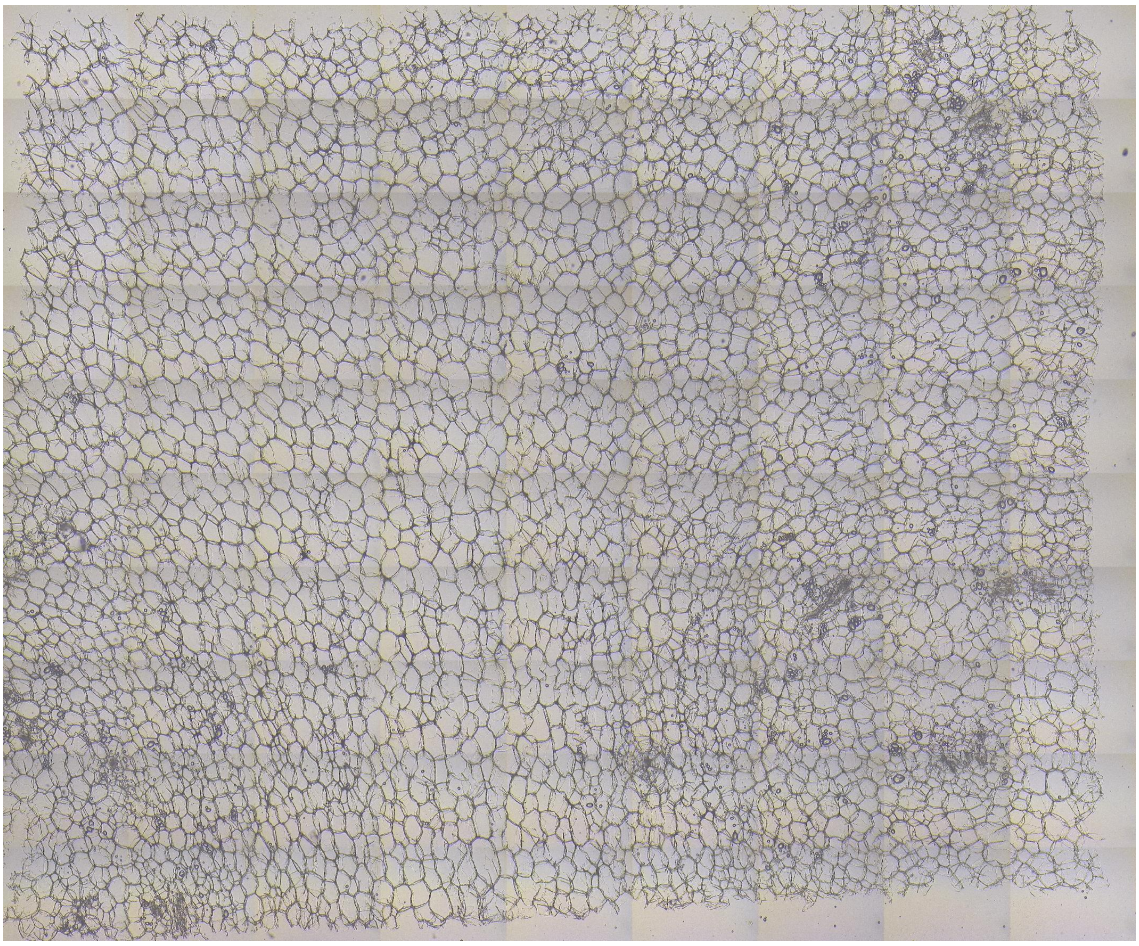


Figure B.3: Original image of figure 5.1c. Named: 151;US;2017;field16;T1;Pith.rep2.jpg

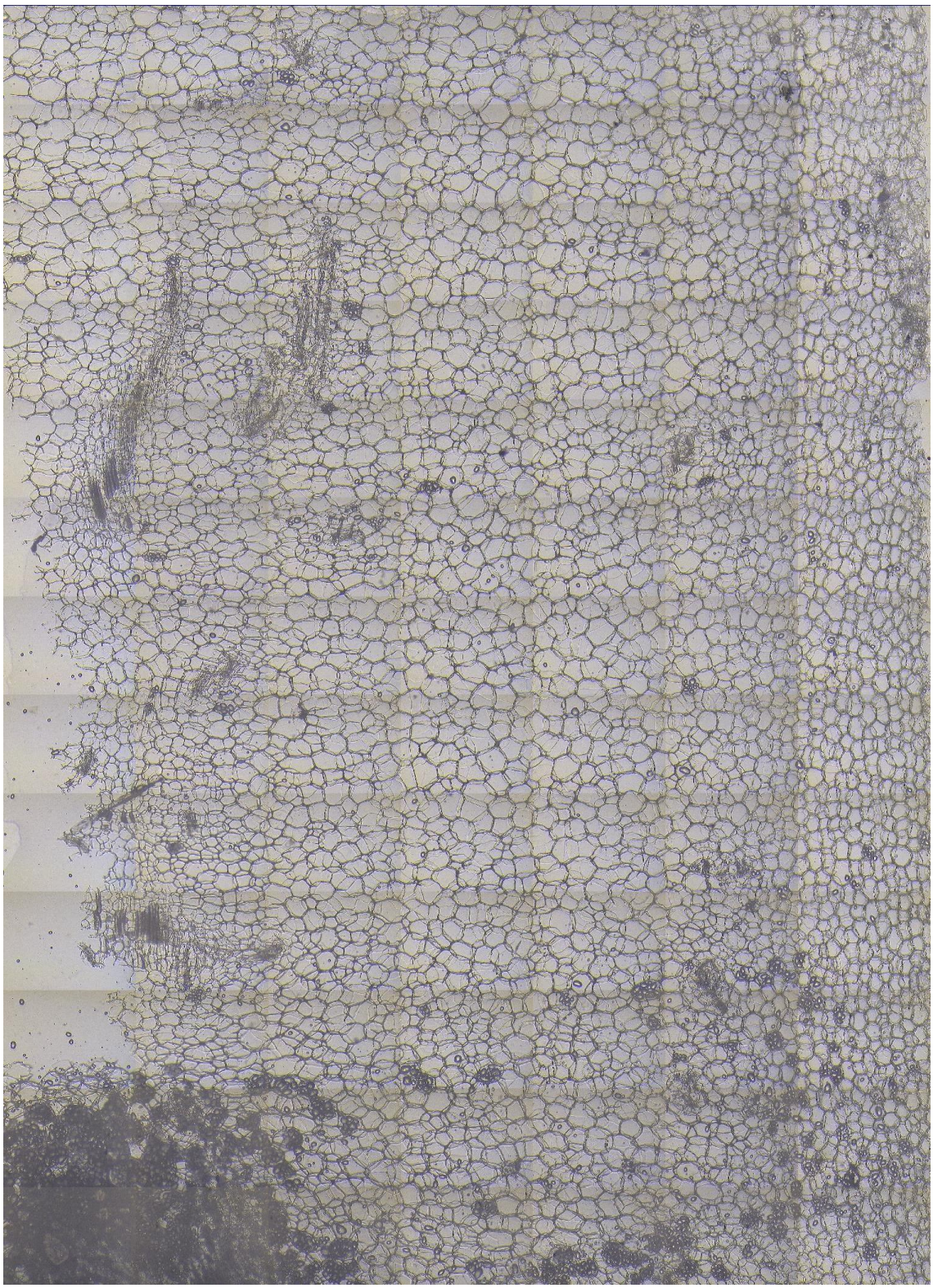


Figure B.4: Original image of figure 5.3a. Named: 11;US;2017;field2;T10;cortex.rep3.jpg

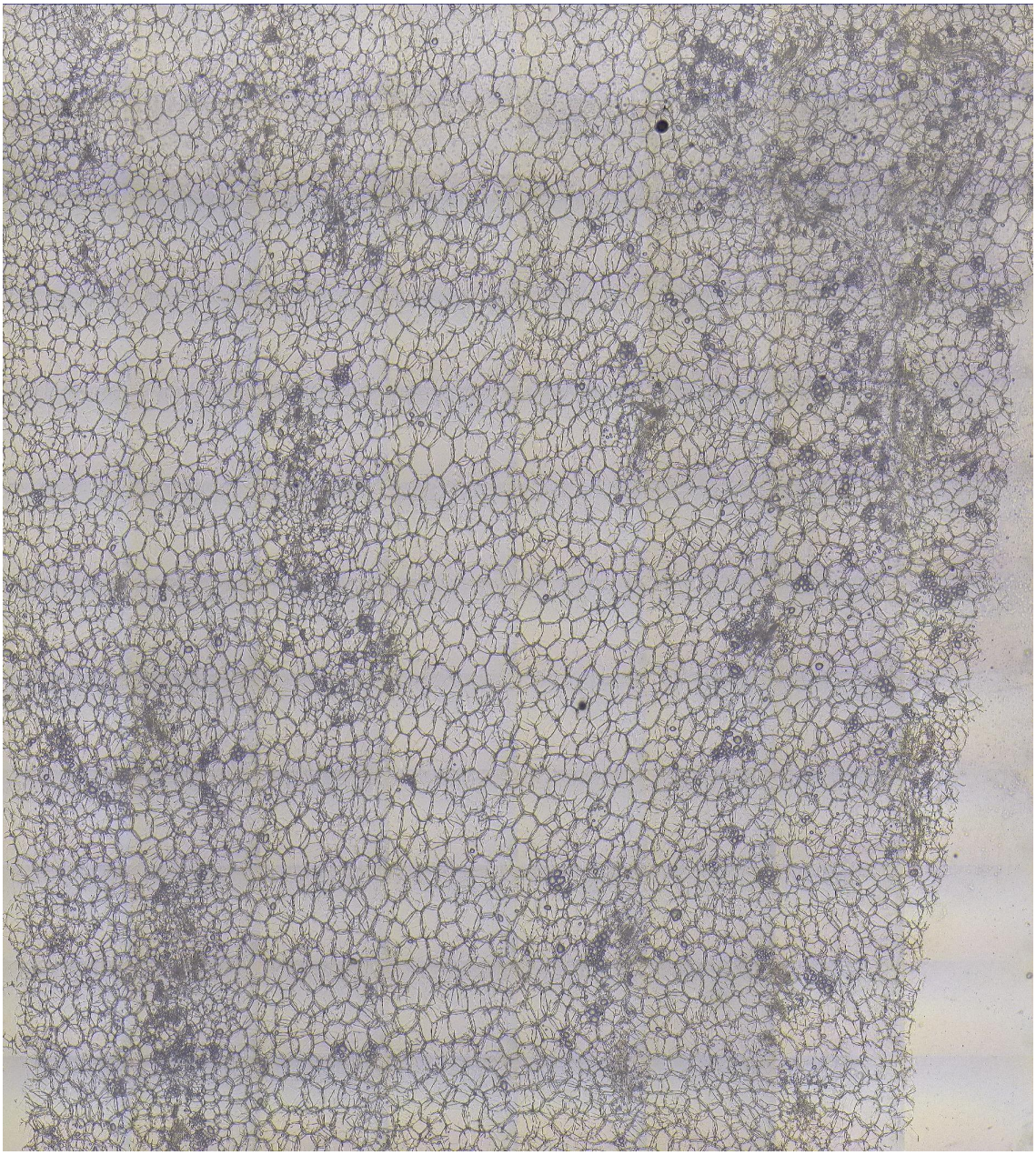


Figure B.5: Original image of figure 5.3b. Named: 11;US;2017;field2;T10;Pith.rep1.jpg

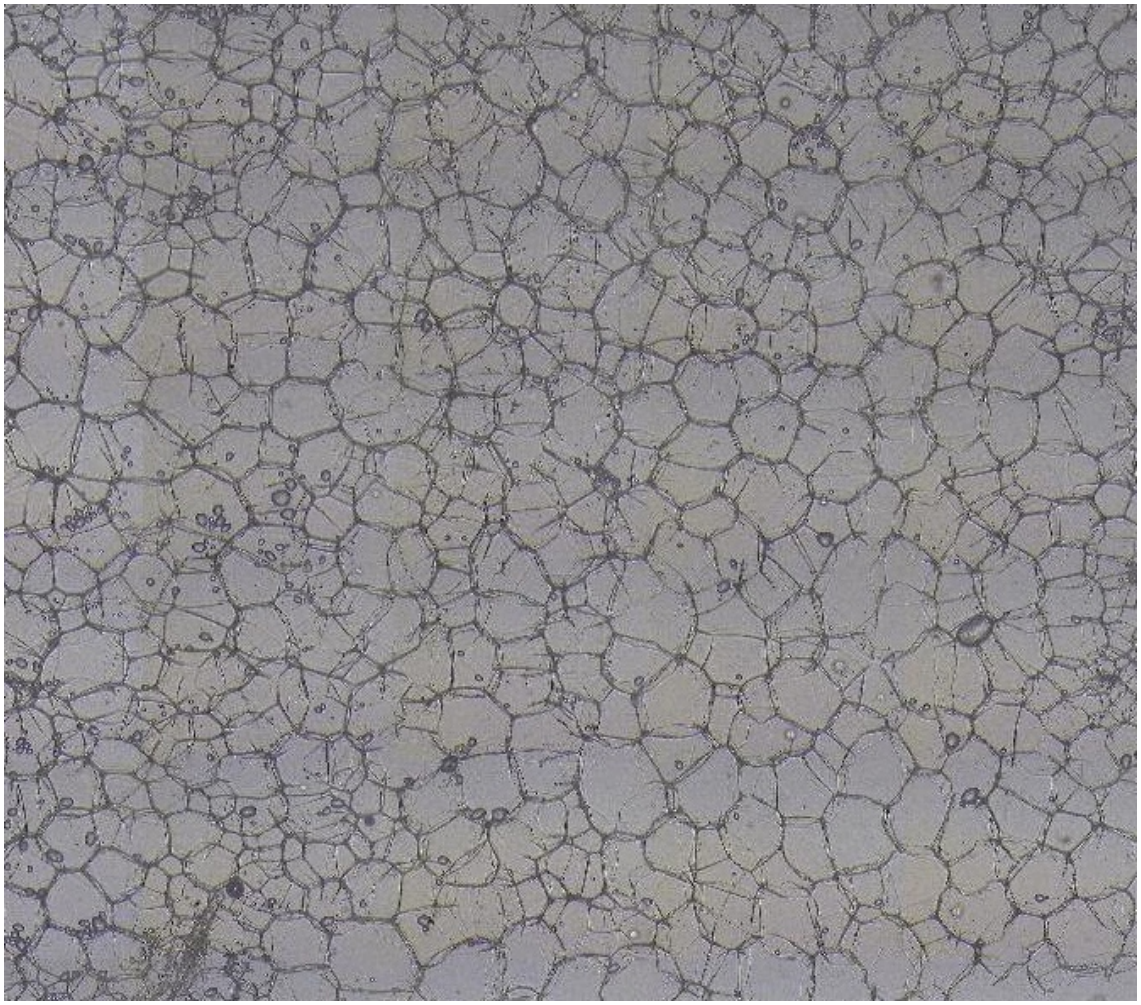


Figure B.6: This image was analysed in section 6.2. Named: pith2.cropped.jpg

C

Practical functions

In this appendix we intend to provide some code. We also provide some technical information on the workings of the cropping algorithm and the resizing algorithm, this as they both are unrelated to the cells themselves. Note that these algorithms are both preprocessing steps in their own right, like we see in Section 2.2.

C.1. Image cropping

Many of the images that we have received are not optimally conditioned, that is around the edges of the images there are many 'white' spaces where there is no potato tissue to be found. In order to prevent false positives in these areas we want to automatically cut off these zones. Hereto we make a simple algorithm to find d, which is the width of the strip that we cut off from the image.

For each edge we perform the following analysis:

1. Initialise the zone width δ at half the image length or width (depending on which edge we consider) to cut the image in two equal parts. And we set $\gamma = 1$, this is the minimal cut off size.
2. Now we calculate the length of the difference of the normalised histograms of both the image parts. We do this for $d = \gamma$ and $d = \gamma + \delta$. We call these values A and B respectively.
3. Now we look at the point which is exactly in the middle between A and B. Again we calculate the length of the difference between the histograms, we call this value M.
4. We calculate the slope between, AB, AM and MB. (to be called dAB, dAM and dMB respectively).

Using these values we can take decisions, whether we include the area between MB in the cropped image, or whether we remove AM entirely. We have the following criteria:

- If $A < B$ or $M < B$, we include MB in the cropped image. We set δ to half its value, γ is unchanged.
- If $dAM > dMB$, we include MB in the cropped image. We set δ to half its value, γ is unchanged.
- If $dAM < dMB$, we exclude AM from the cropped image. γ is set to $\gamma + \delta/2$, then we set δ to half its value.

Then we return to step 2, and repeat until $\delta < 10$. As a step of less than 10 pixels does not improve the result much.
In some sense we can compare this algorithm to an effective way of finding out a number. For instance when we are asked to guess a random value between two limits, all we learn is if our guess is too low or too high (or correct). An example:

We are asked to guess an integer value between 0 and 100. (without our knowing the value is 56).
We guess **50**, we receive **too low**.
Then the new guess is **75**, we receive **too high**.

Then we guess **63**, we again receive **too high**.
Now we guess **57**, which is **too high**.
New guess **54**, **too low**.
then we guess **56**, which is **correct**.

We are not necessarily interested in the precise answer, as our problem is more complicated than just finding some random variable on a scale. So a precision of 10 pixels suffices. We see the result of cropping Figure B.2 in Figure C.1.

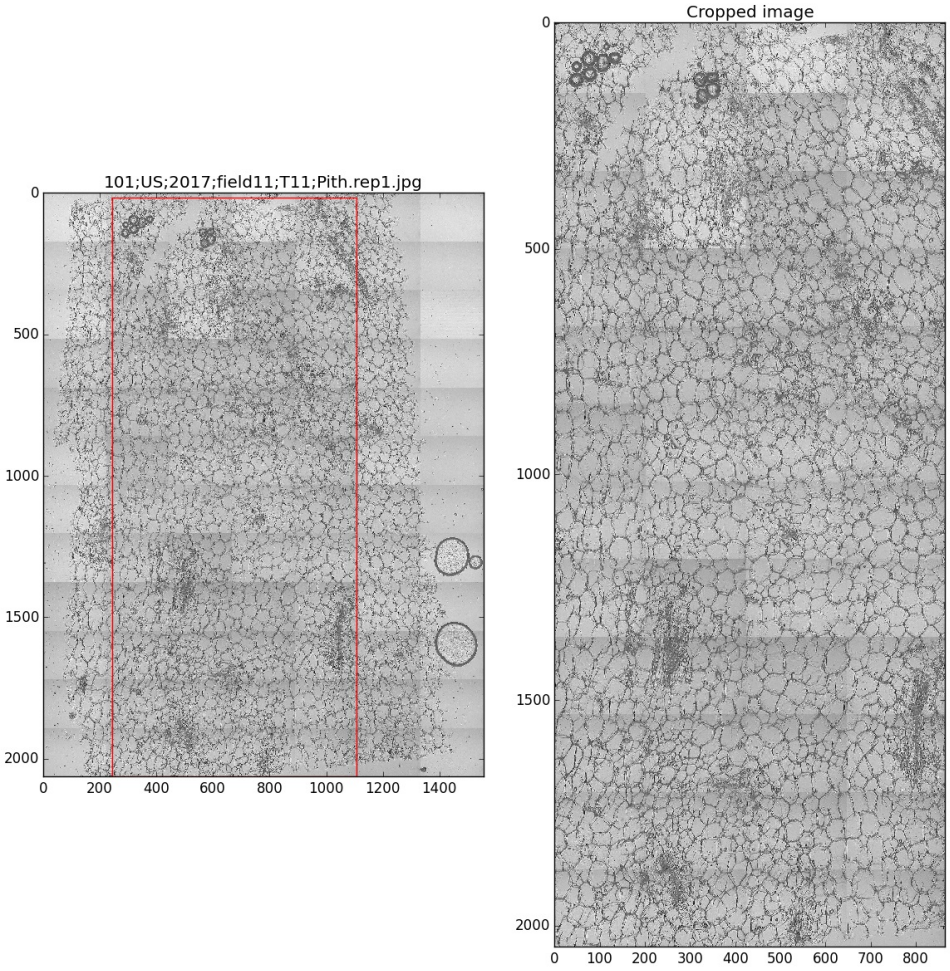


Figure C.1: Here we see the comparison between the original image and the cropped image. Note how the bright edges, in which no potato tissue can be seen, are removed. On the downside, the cropping might be a bit rigorous so some good cells were thrown away too. But less data is preferred over wrong data.

C.2. Image resizing

In order to cope with very large we need to resize the images, this can be done in a very simple manner. Namely we make new 'superpixels', i.e. in the original large image we select tiles of $n \times n$ pixels and take their average brightness and give it to the new pixel. In this way we can speed up the analysis of larger images, but also when we are provided extremely large images. Like for instance high resolution images in which much more pixels are used to display the image. It would be better to use the large image, but if the image contains some 100 million pixels that becomes a computational nightmare. So we reduce the number of pixels greatly, which will allow us to analyse the image.

The code which does this is the following:

```
import numpy as np

def block_view(A, block= (3, 3)):
  #A is the image array and block the new superpixel size
  shape= (A.shape[0]/ block[0], A.shape[1]/ block[1])+ block
  strides= (block[0]* A.strides[0], block[1]* A.strides[1])+ A.strides
  return ast(A, shape= shape, strides= strides)
```

Listing C.1: Image resizing function

The code in Listing C.1 will resize the image. The default superpixel is 3×3 pixels in size. The image is divided in new pixels and these pixels then get their new value, and then the new image is returned which we can save as an array using some saving function.

Bibliography

[1] P.J.W. Iles, *Average Cell Orientation, Eccentricity and Size Estimated from Tissue Images*, Master thesis, University of Waterloo, Waterloo, Ontario, Canada, (2005).

[2] F. Liu, F. Xing & L. Yang, *Robust Muscle Cell Segmentation using Region Selection with Dynamic Programming*, Proceedings IEEE int. Symp. Biomed. Imag., pp. 521, April (2014)

[3] L. Najman & M. Schmitt, *Geodesic Saliency of Watershed Contours and Hierarchical Segmentation*, IEEE transactions on pattern analysis and machine intelligence, vol. 16, no. 12, pp. 1163, (1996).

[4] P. Arbeláez, M. Maire, C. Fowlkess & J. Malik, *Contour Detection and Hierarchical Image Segmentation*, IEEE transactions on pattern analysis and machine intelligence, vol. 33, no. 5, pp. 898, (2011).

[5] M. Kass, A. Witkin & D. Terzopoulos, *Snakes: active contour models*, Int. Journal of computer vision, pp. 321-331, (1988).

[6] Lecture notes, *deformable/active contours (or snakes)*, based on [7, Chp. 4].

[7] E. Trucco & A. Verri, *Introductory techniques for 3-D computer vision*, Upper Saddle River, New Jersey, USA: Prentice-Hall Inc., (1998).
ISBN 0-13-261108-2

[8] M. Sakalli, K. Lam & H. Yan, *A faster converging snake algorithm to locate object boundaries*, IEEE transactions on image processing, vol. 15, no. 5, pp. 1182, (2006).

[9] N.M. Khan & K. Raahemifar, *A novel accelerated greedy snake algorithm for active contours*, IEEE proc. Canadian conference on electrical and computer engineering, Niagara Falls, ON, Canada, pp. 186-190, (2011).

[10] C. Xu & J.L. Prince, *Gradient vector flow: a new external force for snakes*, IEEE Proc. Conf. on Comp. Vis. Patt. Recog., pp. 66-71, (1997).

[11] C. Xu & J.L. Prince, *Snakes, shapes and gradient vector flow*, IEEE transactions on image processing, vol. 7, no. 3, pp. 359-369, (1998)

[12] W. Burger & M.J. Burge, *Digital image processing: an algorithmic introduction using Java*, London, UK: Springer-Verlag London Ltd., (2016). [second edition].
ISBN 978-1-4471-6683-2

[13] *Deep learning for complete beginners: Using convolutional nets to recognise images*, Cambridge Coding Academy.

[14] A.V. Oppenheim, A.S. Willsky & S. Hamid Nawab, *Signals and systems*, Harlow, UK: Pearson education Ltd., (2014). [second edition].
ISBN 978-1-292-02590-2

[15] J.J.I.M. van Kan, A. Segal & F.J. Vermolen, *Numerical methods in scientific computing*, Delft, Zuid-Holland, Netherlands: Delft Academic Press, (2014) [second ed.].
ISBN 97890-6562-3638

[16] B. de Pagter & W.G.M. Groenevelt, *Dictaat WI1601, analyse 2*, Delft, Zuid-Holland, Netherlands: TU Delft, (2012) [ed. 2012-2013].

[17] A.W. Fitzgibbon, M. Pilu & R.B. Fisher, *Direct least squares fitting of ellipses*, Proc. of the 13th international conf. on pattern recognition, pp. 253-257, Vienna, (1996).

[18] A.W. Fitzgibbon, M. Pilu & R.B. Fisher, *Direct least squares fitting of ellipses*, Pattern analysis and machine intelligence, vol. 21, no. 5 , pp. 476, (1999).

[19] A. Lipson, S.G. Lipson & H. Lipson, *Optical physics*, Cambridge, UK: Cambridge university press, (2011). [fourth edition].
ISBN 978-0-521-49345-1

[20] K. de Clerk & T.S. Buys, *Analytical efficiency in chromatography. I. Qualitative efficiency*, Separation science, vol. 7, no. 4, pp. 371-387, (1972).

[21] R. Shinnar & G.H. Weiss, *A note on the resolution of two Gaussian peaks*, Separation science, vol. 11, no. 4, pp. 377-383, (1976).

[22] R.A. Adams & C. Essex, *Calculus: a complete course*, Toronto, Ontario, Canada: Pearson Canada Inc., (2010). [seventh edition].
ISBN 978-0-321-54928-0

[23] D.J. Griffiths, *Introduction to electrodynamics*, Glenview, Il, USA: Pearson Education Inc. (2013). [fourth edition].
ISBN 978-0-321-84781-2

[24] C. Vuik & D.J.P. Lahaye, *Lecture notes: Scientific computing*, Delft, Zuid-Holland, Netherlands: TU Delft, (2015).

[25] J.A. Rice, *Mathematical statistics and data analysis*, Belmont, CA, USA: Brooks/Cole, (2007) [third int'l ed.].
ISBN 978-0-495-11868-8

[26] Michael Breuß, Alfred Bruckstein & Petros Maragos, *Innovations for shape analysis: models and algorithms*, Heidelberg, Germany: Springer-Verlag, (2013).
ISBN 978-3-642-34140-3