Delft University of Technology
Faculty of Electrical Engineering, Mathematics & Computer Science
Delft Institute of Applied Mathematics

Extraction of biological parameters of wound-healing processes from time-lapse videos

Extractie van biologische parameters van wondhelende processen uit time-lapse video's

Bachelor thesis submitted to the
Delft Institute of Applied Mathematics
as part to obtain

the degree of

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

by

DANIEL TSANG

Delft, The Netherlands
June 2019

TU Delft

BSc thesis APPLIED MATHEMATICS

"Extraction of biological parameters of wound-healing processes from time-lapse videos"

"Extractie van biologische parameters van wondhelende processen uit time-lapse video's"

DANIEL TSANG

Delft University of Technology

Supervisors

Dr. N.V. Budko
Dr.ir. F.J. Vermolen

Other members of the committee

Drs. E.M. van Elderen

June, 2019 Delft

1 Abstract

Experimental data have been extracted from wound-healing time-lapse videos. These data include the detection of the area of the wound and the detection of individual cells. Extracting the area of the wound is done by using a function in Python `OpenCV` called `cv2.findcontours`. An improved method to extract the area of the wound is introduced as well using the Sobel filter. Detecting the individual cells is done by localizing the local maxima in an image. Histogram equalization is applied to enhance the global contrast to increase the performance of cell detection. The continuum model is applied in one of the videos, where we used different parameters to model the data. A method is described to find the optimal parameter of the continuum model by using the method of least-squares. Finally, two methods using the kernel density estimation are described which can be useful in future studies.

Contents

2 Introduction

The mathematical modelling of wound-healing processes is an active research field, where many people have already been working on for many years in this research area, such as [3] and [9]. Many mathematical models have been developed in order to describe such a complicated biological process. For example in [3], a mathematical model for tissue regeneration has been considered, which can be applied to bone regeneration as well. This model is based on the reaction-diffusion equation, which describes the rate of change of the concentration of a certain substance. In this paper, the concentration of growth factors was examined. These factors play an important role in the regeneration of bones and other tissues.

Another example is described in [9]. Here, a model of epidermal[1] wound healing has been developed and investigated. The model described in this paper is also based on the reaction-diffusion equation, but now the cell density per unit area was taken into consideration. The last example is the so-called 'Cellular Automata' model described in [6]. This paper formulated a spatial Markov-Chain model to describe the progression of skin cancer. This model is based on a probabilistic point of view, where cells are either in 'cancer-state' of in 'non-cancer state', both with a certain probability. Although the paper used this model to simulate the progression of skin cancer, there is the possibility to adjust or to expand this model to describe wound-healing processes as well.

Eventually, all these mathematical models that have been developed over the years must be evaluated against experimental data, to see how accurately certain models describe such a complicated process. One way to retrieve experimental data is by looking at wound-healing videos on the internet, such as on YouTube. The aim of this project is to extract data or parameters from these videos to evaluate it against existing mathematical wound-healing models. The videos we are going to use are all recorded after performing an in vitro scratch assay, which is a method used to study cell-migration in vitro[2]. The links of the videos being used are found throughout the thesis.

In this project, the first parameter we are going to extract from these videos is the area of the wound and how it changes over time. This is described in section 3 and section 6. After that, we are going to measure the amount of cells in a given video(frame), from which we can determine the cell density. This will be discussed in section 4 and section 6. After that, we are going to fit one of the existing wound-healing models to the data we extracted from the videos. This is described in section 5. At last, we discuss two methods we wanted to use to extract data from videos as well, but further research and improvements are needed to use these methods correctly.

[1] The skin consists of three layers, where the epidermis is the uppermost layer of those three.
[2] meaning: *in the glass.*

3 Frame processing

In our first goal to extract the wound-healing data from a video, we have to analyse the video itself at first. Throughout the frame processing, we are going to use Python 3.7.3 and open-source libraries such as `OpenCV`, which stands for `Open source Computer Vision`. This library is specifically developed to analyse images and videos with many built-in functions which we will use.

3.1 Extracting the area of the wound

One of the things we want to extract from a video is the change of the area of the wound over time. Under normal circumstances, we expect that the area of the wound declines over time and eventually the wound vanishes. In our attempt to extract the area of the wound from the video and looking at its change over time, we will take the following steps:

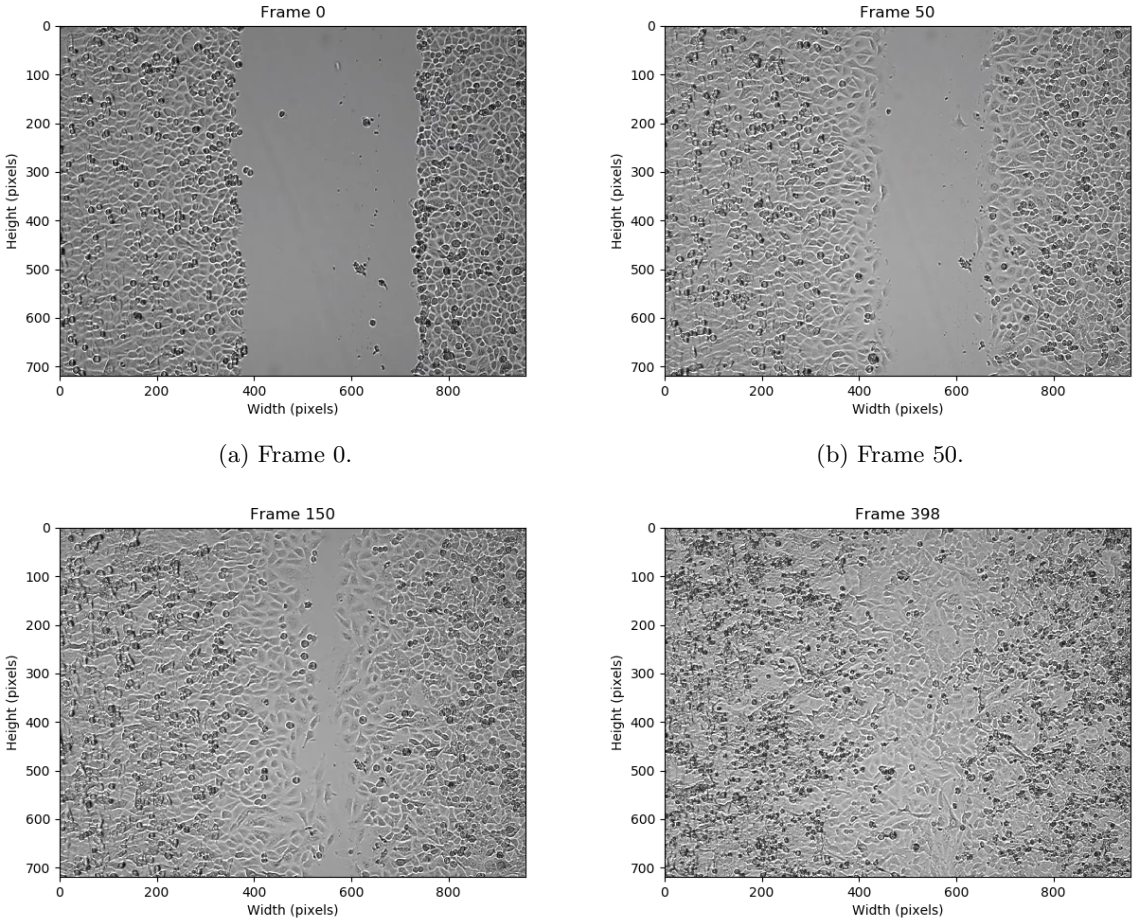
1. First, we will divide the video into a collection of frames.
2. After that, every frame will be analysed. This means that every frame will be filtered in order to distinguish the wound from its surrounding.
3. Next, when the wound is distinguished from its surrounding, we draw a contour that encloses the wound.
4. And finally, we calculate the area of the wound that is enclosed by the contour and repeat this process for every frame.

3.1.1 Frames

The first video[3] we are going to analyse shows us a clear distinction between the wound and the tissue on the sides. Because of this, we found that this is a good video to start with. This video lasts for 39 seconds and with a built-in function in `OpenCV`, the video is then divided into 399 frames.

On the next page in Figure 1 there are four frames of the video displayed. The sizes of the frames are in pixels, with height 720 pixels on the vertical axis and width 960 pixels on the horizontal axis. As you can see, the wound becomes smaller because the cells on the sides are moving towards the wound, and eventually, the gap is closed.

[3] https://www.youtube.com/watch?v=v9xq_GiRXeE



(a) Frame 0.

(b) Frame 50.

(c) Frame 150.

(d) Frame 398.

Figure 1: Several frames of the video displayed.

3.1.2 Analysis of the frames

In the next step, we need to distinguish the wound from its surrounding, so that we are able to detect the boundary of the wound, which will be explained later on in the next section. Distinguishing the wound can be done by making a binary image out of the frame using a suitable threshold for the intensity values of the wound. The idea then is to choose a value such that the wound in the binary image will have the color white, whereas the color outside the wound will be black.

Before we determine what threshold value we should choose, we need to convert the frame into a grayscale image. A grayscale image is an image in which the colors are composed exclusively of shades of gray. In this way, all the pixels will have intensity values that ranges from 0 to 255, with 0 black colored and 255 white colored, so that our threshold value lies in this range as well.

After converting the image, we can start with the determination of a suitable threshold value so that we can distinguish the wound from its surrounding.

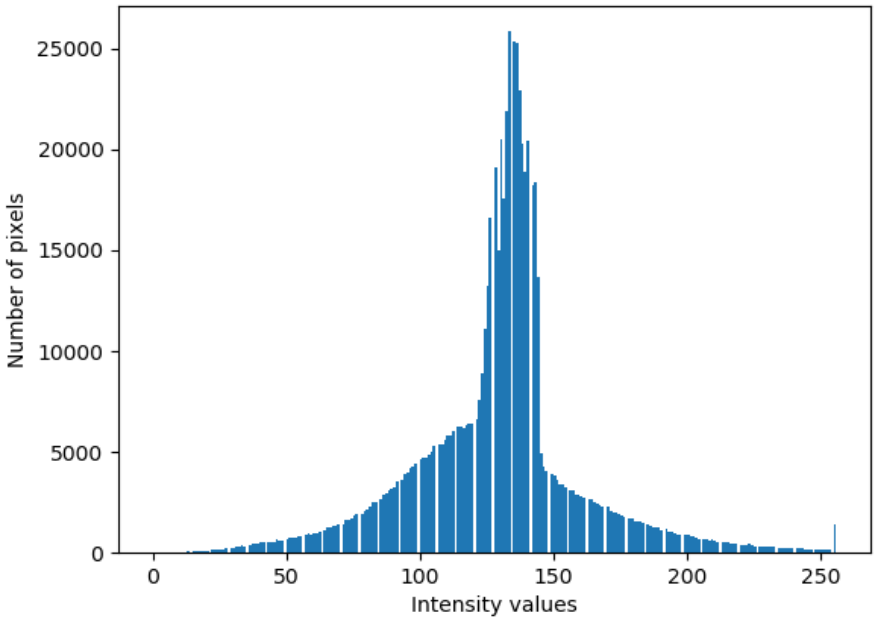
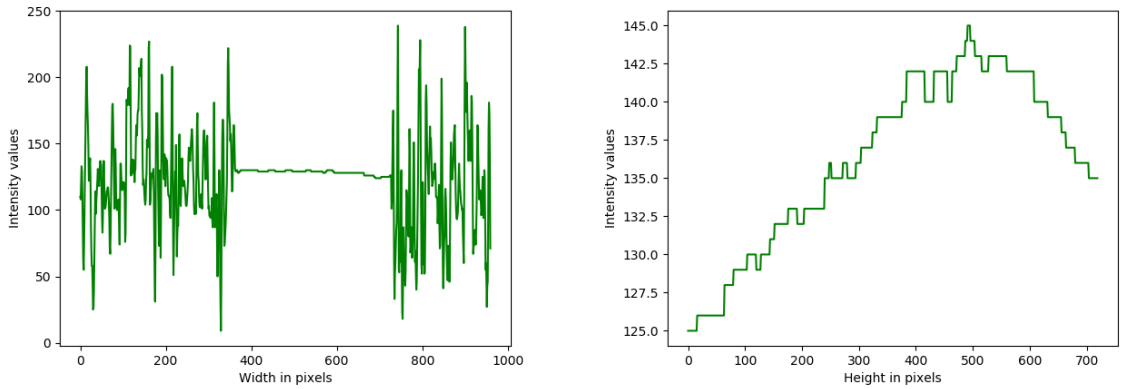


Figure 2: Intensity histogram of the frame.

First, we will look at the intensity histogram of the frame, in particular the histogram of frame 0. This is shown in Figure 2. We see in this histogram the intensity values on the horizontal line and the number of pixels with those intensity values on the vertical line. Clearly, most of the pixels correspond to the intensity values between about 120 and 150. This is also not surprising, because this corresponds to the wound seen in the frame. Overall, the wound has a more or less the same (monotone) colour, which means that the intensity values are in the same range. Moreover, the size of the wound is quite large compared to the size of the frame. Thus the number of pixels with values in that specific range must be very high as well, which explains the large spike in the histogram.

To confirm this even more, we make two plots to see how the intensity values look like if we plot the values along the horizontal and vertical line.



(a) Intensity along the horizontal line.

(b) Intensity along the vertical line.

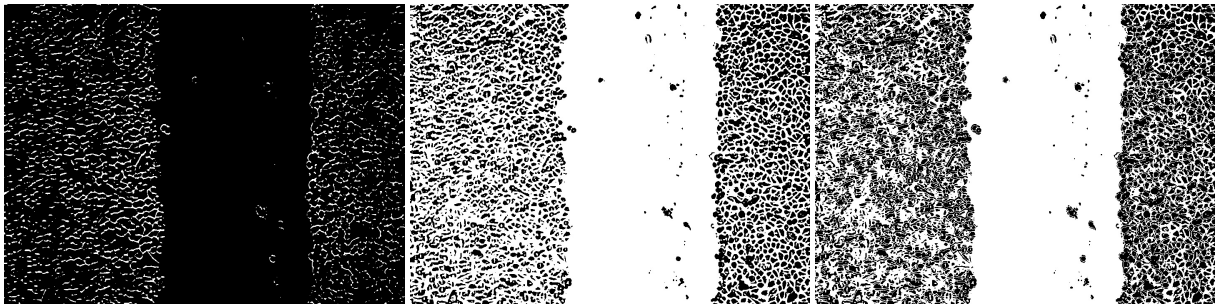
Figure 3: Intensity line in one dimension.

In Figure 3a, we fixed a value on the vertical (height) line and then we plot the intensity values over that particular line. In this case, we chose height = 100 pixels. But, if we choose another height, then

the plot will be similar to that of Figure 3a. In this plot, we see that there is an almost horizontal line from about width 380 to 700 pixels. If we look back at Figure 1a, it is not difficult to see that this line corresponds to the intensity value of the wound, at height = 100 pixels. This value is around 130.

In Figure 3b, we did the same as in Figure 3a, but now we fixed a value on the horizontal (width) line. For this plot, we chose width = 500 pixels, so that the line and the corresponding intensity values are only from the wound itself. From the plot we see that the values are between 125 and 145.

We now know that the wound has intensity values in the range of 120 to 150, so we actually want that those values will have the same colour (white) and all other intensities black. So we put two thresholds on the image separately, with values 150 and 120. After that we subtract the images from each other to receive the binary image. This can be seen in Figure 4.



(a) Threshold value 150. (b) Threshold value 120. (c) Subtracted image.

Figure 4: Results of putting thresholds on the image.

Although there are some black spots, the subtracted image in Figure 4c shows us the white-coloured wound. But outside the wound, there are still some white-coloured spaces, which we do not want because otherwise too many contours are drawn. This will be explained in 3.1.3. In order to remove the white spaces outside the wound, we use the function `cv2.erode` from `OpenCV`. This function is based on the morphological operation erosion. The effect of this operation is that it erodes away the boundaries of regions of white pixels, so that these regions will shrink in size, and holes within those regions become larger.

To use this function, it needs two data sets as input. The first one is the input image where we apply this operation. The second one is called a *structuring element* or *kernel*. The kernel is a small set of coordinate points that determines the precision of this operation on the input image. Mathematically, the definition of erosion for binary images is as follows:

Definition 3.1. *Let $E = \mathbb{R}^2$ be the Euclidean space and let X be the set of Euclidean coordinates corresponding to the input binary image. Furthermore, let K be the set of coordinate points corresponding to the kernel. Then* ***erosion*** *of the binary image X by the kernel K is defined by:*

$$X \ominus K = \{x \in E | K_x \subseteq X\},$$

where K_x is the translation of K by the vector x, so that the center of K is at x, or:
$K_x = \{k + x | k \in K\}, \forall x \in E.$

The result of eroding the image in Figure 4c can be seen in Figure 5.

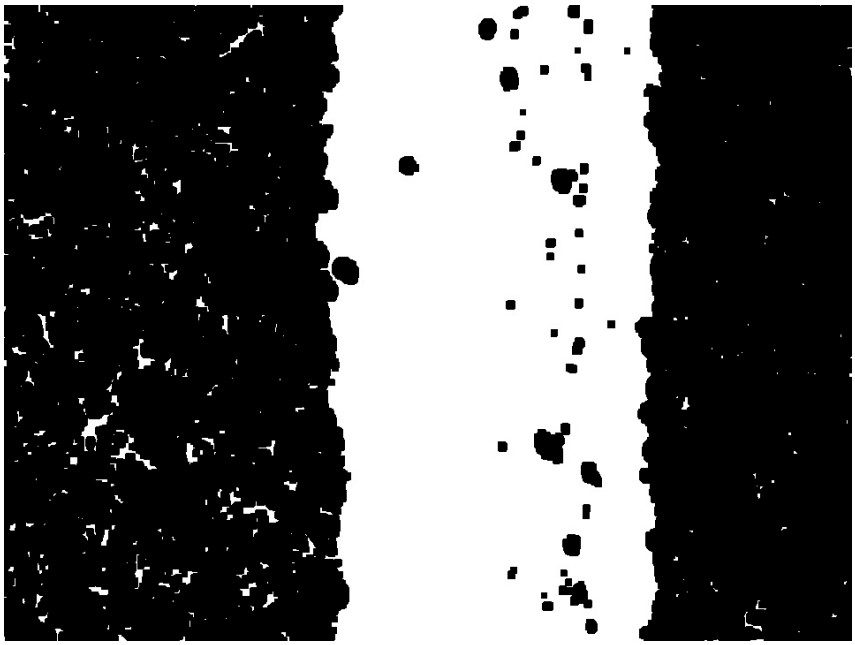


Figure 5: Erosion of the image.

3.1.3 Contour drawing

Now that we have eroded the image, we are able to find and draw intensity levels contours in the image. Again, we use a function in `OpenCV` that finds the contour for us. This function is called `cv2.findContours`, which uses an algorithm that is explained by Satoshi Suzuki et al [1]. In short, this algorithm scans every pixel in a given binary image and it decides whether a pixel is a border point or not. A border point is defined as a 1-pixel (white) (i, j) having a 0-pixel (black) (p, q) in its 8-(4-) neighborhood. The result of this algorithm is that it finds all white objects from a black background in a binary image and calculates the contours around those objects. That is why we had to make sure the wound is white-colored after putting a threshold on the frame and eroding the image to have a better accuracy of finding the contour of the wound.

The function `cv2.findContours` returns a list of all the contours that the function finds and every contour is a `Numpy` array of coordinates (x, y) of boundary points of the contour. Now to find the contour of the wound, we assume that this contour is the largest of all other contours that the function may find. We make this assumption because in Figure 5 we see that the wound is indeed the largest white object after erosion. Thus, the contour with an array length that is larger than any other contour will be chosen as the wound contour. After that, we use the function `cv2.drawContours` to draw the wound contour. The result of the contour is given in Figure 6.

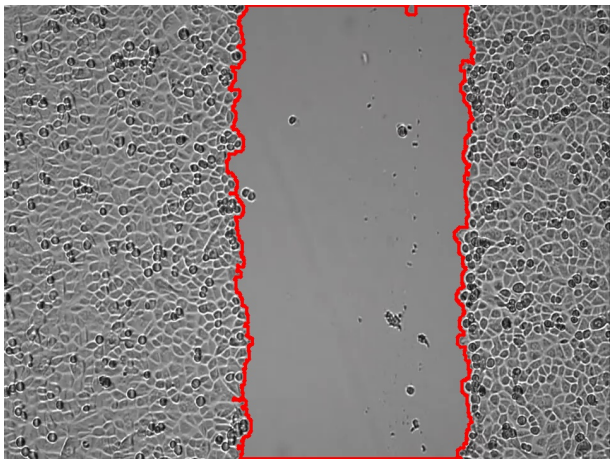
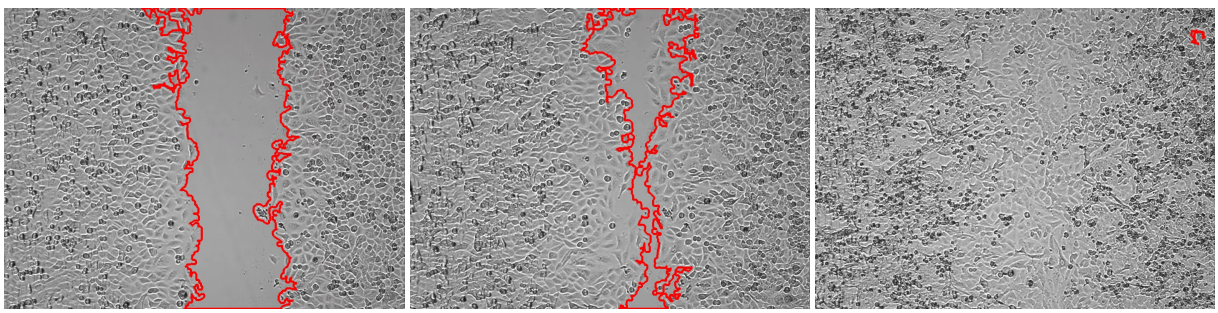


Figure 6: Contour of the wound.

As we can see, the contour is drawn quite accurately because it follows the edges of the wound, and so the contour encloses the whole wound. If we apply the threshold, erode and then calculate the contour of the wound for the other frames displayed in Figure 1, we will get the following contours in Figure 7.



(a) Frame 50. (b) Frame 150. (c) Frame 398.

Figure 7: Contours drawn in other frames.

3.2 Area calculation

In the previous section, we were able to find and draw the contour of the wound for every frame. These contours all consist of a finite amount of pixels (x, y). Now in order to calculate the (approximate) area of the wound encircled by the contour, we can make use of *Green's Theorem*. This theorem states as follows:

Theorem 3.2 (Green's Theorem)**.** *Let P and Q be continuous functions of (x, y) with continuous partial derivatives*

$$\frac{\partial P(x,y)}{\partial y}, \qquad \frac{\partial Q(x,y)}{\partial x}$$

and let C be a piecewise smooth, positively oriented, simple closed curve in a plane. Furthermore, let D be the region bounded by C. Then the following holds:

$$\oint_C P(x,y)\,dx + Q(x,y)\,dy = \iint_D \left(\frac{\partial Q(x,y)}{\partial x} - \frac{\partial P(x,y)}{\partial y}\right) dx\,dy. \tag{3.1}$$

Now from Calculus we know that the area of the region D is equal to $\iint_D dx\,dy$. So if we use Green's Theorem to calculate the area of D, we have to set:

$$\frac{\partial Q(x,y)}{\partial x} - \frac{\partial P(x,y)}{\partial y}\,dx\,dy = 1. \tag{3.2}$$

There are many ways to define P and Q such that the integrand is equal to 1. By choosing $P(x, y) = 0$ and $Q(x, y) = x$, it is easy to see that the requirement is satisfied. So we have that the area A of the region D is equal to:

$$A = \oint_C x\,dy. \tag{3.3}$$

Now in our case, the curve C around the wound consists of many small line segments, so we can write $C = C_0 \cup C_1 \cup \ldots C_{n-1} \cup C_n$, where C_k starts at the point (x_k, y_k) and ends at the point (x_{k+1}, y_{k+1}). From Calculus we know that for a line integral evaluated over a piecewise smooth curve, the integral is equal to the sum of the integrals evaluated over each of the pieces:

$$A = \oint_C x\,dy = \int_{C_0} x\,dy + \int_{C_1} x\,dy + \ldots + \int_{C_n} x\,dy \tag{3.4}$$

In order to compute the integral over a line segment C_k, we have to parametrize the line segment from (x_k, y_k) to (x_{k+1}, y_{k+1}). We set $x(t) = (x_{k+1} - x_k)t + x_k$ and $y(t) = (y_{k+1} - y_k)t + y_k$, with $0 \leq t \leq 1$. If we substitute the parametrization into the integral, we get the following:

$$\begin{aligned}\int_{C_k} x\,dy &= \int_0^1 ((x_{k+1} - x_k)t + x_k)(y_{k+1} - y_k)\,dt \\ &= \frac{(x_{k+1} + x_k)(y_{k+1} - y_k)}{2}\end{aligned}$$

By summing over all the line segments C_k's, we get the total area A of the region D:

$$A = \sum_{k=0}^{n} \frac{(x_{k+1} + x_k)(y_{k+1} - y_k)}{2} \tag{3.5}$$

We can now implement equation (3.5) in Python to calculate the area of the wound by calculating the area of the contour in pixels. We have done this for every frame, so we get the following result in Figure 8.

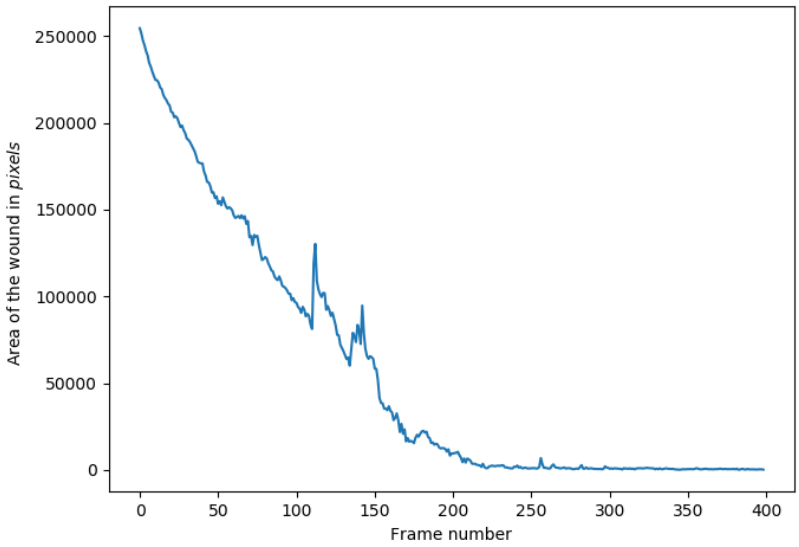


Figure 8: Area of the wound per frame.

The result in Figure 8 shows us that the area indeed decreases as the time (frames) passes by and eventually the wound is closed. But surprisingly, we see two spikes between the frames 100 and 150. Unless something happens unexpectedly, the spikes should not be there at all, because we see in the video that the area decreases constantly. If we zoom in on the spikes, we see that the sudden increase is at frame 112 and frame 142 in Figure 9. To see what is happening with the contours, we will look at the frames around 112, because that spike is larger than the second one, so that we have a better look at what is happening.

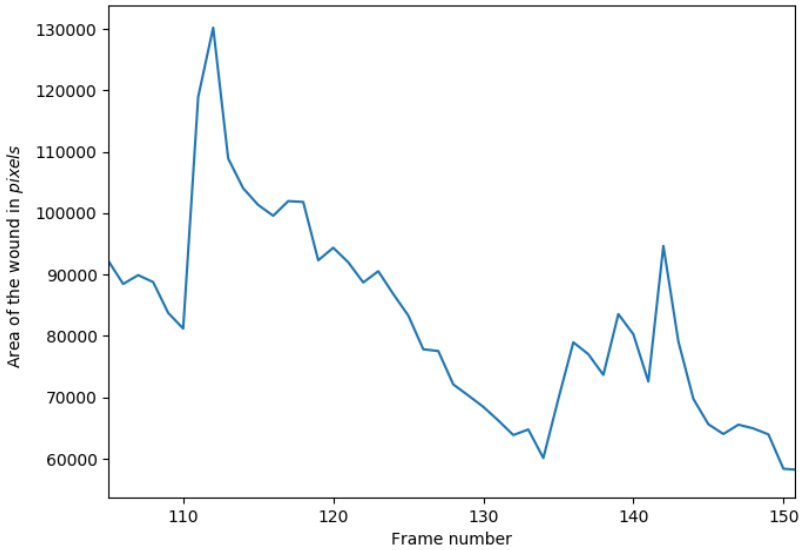
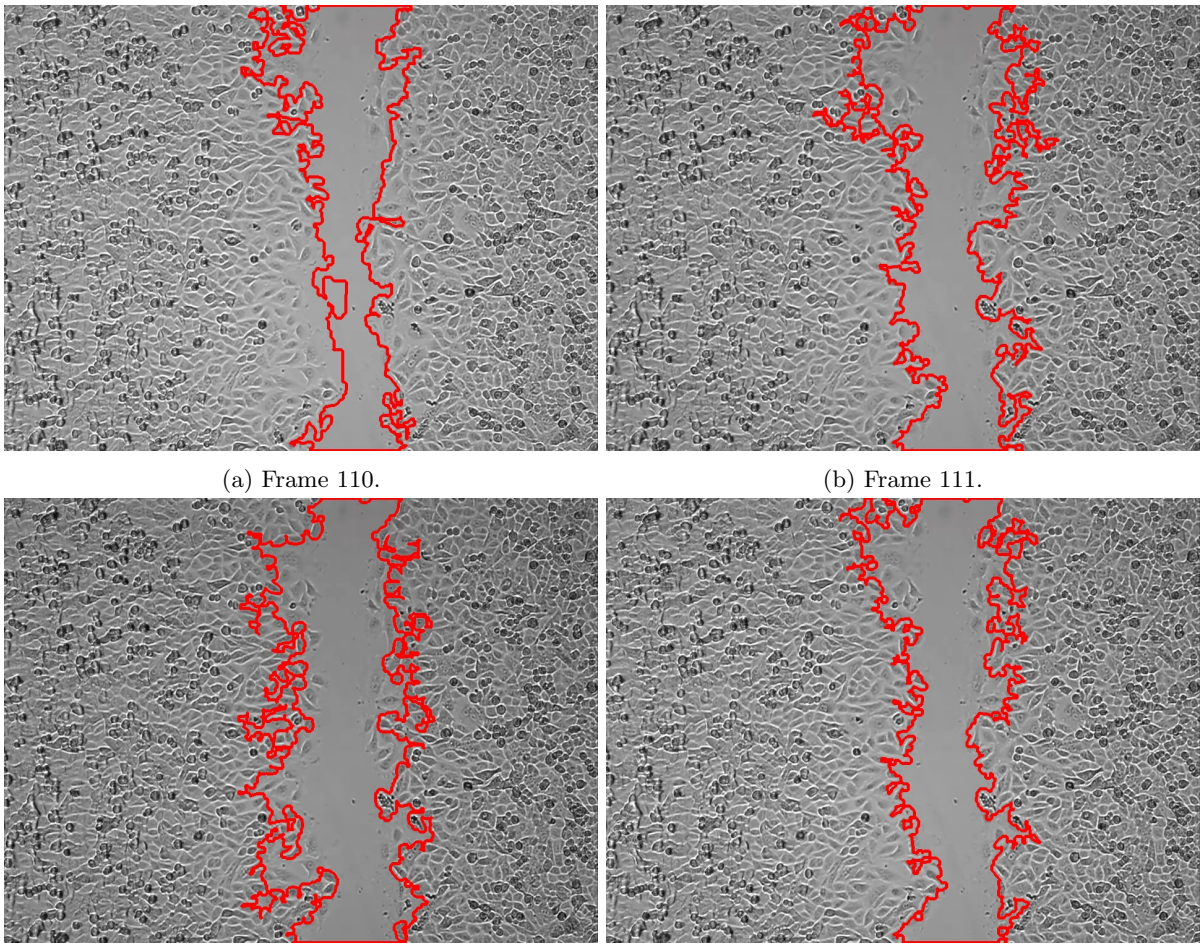


Figure 9: Close-up on the spikes in Figure 8.

We can see in Figure 10b and Figure 10c that the sudden increase is due to the fact that the contour is partially drawn outside the wound. So the area of the contour increases because now both the wound and a part of the area outside of the wound are taken into account. Also, comparing to Figure 6, these contours in Figure 10 contain some small loops which affect the area calculation as well. So there are some improvements needed to have a better approximation of the area of the wound.



(a) Frame 110. (b) Frame 111.

(c) Frame 112. (d) Frame 113.

Figure 10: Frames around frame 112.

3.3 Improvements on extracting the area of the wound.

As we have seen in the previous sections, we were able to detect the wound and calculate the area of it. However, we also saw that by detecting the wound, the contour that is drawn is not very accurate. It has some loops and part of the cells just outside the wound are also included in the contour. So to have a better estimation of the area of the wound, we need a different method to detect the wound.

3.3.1 Sobel Operator

The Sobel operator (or Sobel filter) is an operator that is used in image processing, in particular to detect edges within an image. The operator calculates the gradient of the intensity function of the image. This is very useful in edge detection, because finding an edge in the image is to find the regions in the image where the intensity changes suddenly (an increase or decrease), see for example again in Figure 3a. If we can calculate the gradient (derivative) of the sudden change in intensity value of a certain pixel, then we know if that pixel belongs to an edge or not. In other words, in order to detect edges in an image, we need to locate the pixel locations where the gradient is higher than the gradient of its neighbours.

The Sobel operator actually calculates two derivatives. Namely the change of intensity in the horizontal line and in the vertical line. This is done by convolving the input image with the kernels S_x for the

horizontal derivative and S_y for the vertical derivative. The convolution is defined as:

$$g(x,y) = k * I(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} k(s,t) I(x-s, y-t), \tag{3.6}$$

where $g(x,y)$ is the output image after convolution, k is the kernel being used and I the input image. If the kernel is of size 3×3, then s and t ranges from -1 to 1.

If we define I as the input image, G_x and G_y the output images containing the horizontal and vertical derivative at each point respectively, then the operation is defined as:

$$G_x = S_x * I = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I \tag{3.7}$$

for the horizontal derivative and

$$G_y = S_y * I = \begin{bmatrix} +1 & 2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * I \tag{3.8}$$

for the vertical derivative, where * here is the convolution operation.

The kernels S_x and S_y can be decomposed as a product of two smaller kernels. The decomposition is given by

$$S_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} = \begin{bmatrix} +1 \\ +2 \\ +1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & +1 \end{bmatrix}, \tag{3.9}$$

and

$$S_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ +1 \end{bmatrix} * \begin{bmatrix} +1 & +2 & +1 \end{bmatrix}. \tag{3.10}$$

Note that the kernels $[-1, 0, +1]$ in S_x and $[-1, 0, +1]^\top$ in S_y are the coefficients of the numerical approximation of the derivative of the intensity function.

Remember that the derivative of a function f (assuming all the conditions are satisfied) at a point x is defined by the limit

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}. \tag{3.11}$$

To approximate the derivative, we can make use of finite differences. There are three methods of these which are used most often:

- *Forward difference*, defined by

$$Q_f = \frac{f(x+h) - f(x)}{h}, \quad h > 0.$$

- *Backward difference*, defined by

$$Q_b = \frac{f(x) - f(x-h)}{h}, \quad h > 0.$$

- *Central difference*, defined by

$$Q_c = \frac{f(x+h) - f(x-h)}{2h}, \quad h > 0.$$

So when h is small, it would approximate the derivative of the function f in a point x. However, we know from Numerical Methods that the approximation error of the forward and backward differences are $\mathcal{O}(h)$, which tends to zero if $h \to 0$, whereas for the central difference, the approximation error is $\mathcal{O}(h^2)$, which tends faster to zero if $h \to 0$. So the central difference gives us a better approximation of the derivative as $h \to 0$. The Sobel operator uses the central difference method to calculate the derivative of the intensity function at a certain pixel. Notice that the kernel $[-1, 0, +1]$ represents the central difference, with -1 the pixel at $x - h$ and 1 the pixel at $x + h$ of x. The kernels $[+1, +2, +1]$ and $[+1, +2, +1]^\top$ represent the (weighted) averaging kernels. To illustrate how the operator works, we will show a quick example of this operation.

Example 3.3. *Assume we have an input image I, which is grayscaled and the kernel S_x defined by*

$$I = \begin{bmatrix} 50 & 50 & 200 & 200 \\ 50 & 50 & 200 & 200 \\ 50 & 50 & 200 & 200 \\ 50 & 50 & 200 & 200 \end{bmatrix}, \quad S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}. \tag{3.12}$$

As we can see, between the second and third column, the intensity difference between the values in those columns is large. This indicates that we are dealing with an edge. Now the convolution is done by putting the kernel S_x over the elements with the same colour. Then calculating the sum of all the elements that have been weighted, we have -50 + -100 + -50 + 200 + 400 + 200 = 600. This value is non-zero, so there is an edge. If the kernel was placed over a region of an image where no edges can be seen (for example when all the elements have the same value), then the sum would be zero. So the brighter the edge, the bigger the value of the sum. Note that the sign of the value does not matter whether or not an edge is detected.

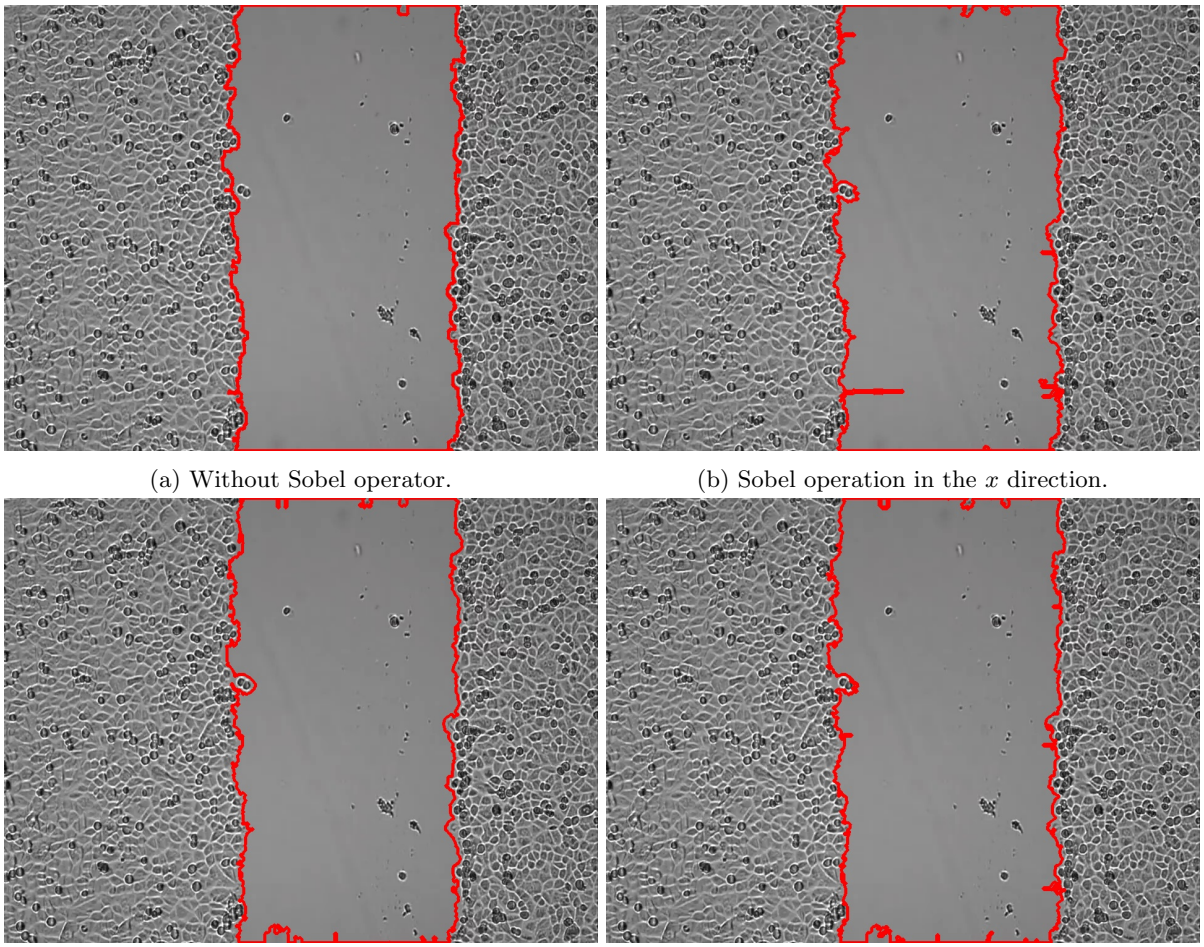
As we said earlier, the Sobel operator computes the gradient in the x direction and the gradient in the y direction at each point. But it is also possible to combine these two approximations to calculate the magnitude of the gradient in each point. This can be done using:

$$G = \sqrt{G_x^2 + G_y^2}. \tag{3.13}$$

This is also called the *gradient magnitude*. The following equation can be used as well:

$$G = |G_x| + |G_y|. \tag{3.14}$$

Now we will look at some of the results using the Sobel operator. First, we will compare the contours that have been drawn on the frames using the Sobel operator and without the operator, see Figure 11.



(a) Without Sobel operator.

(b) Sobel operation in the x direction.

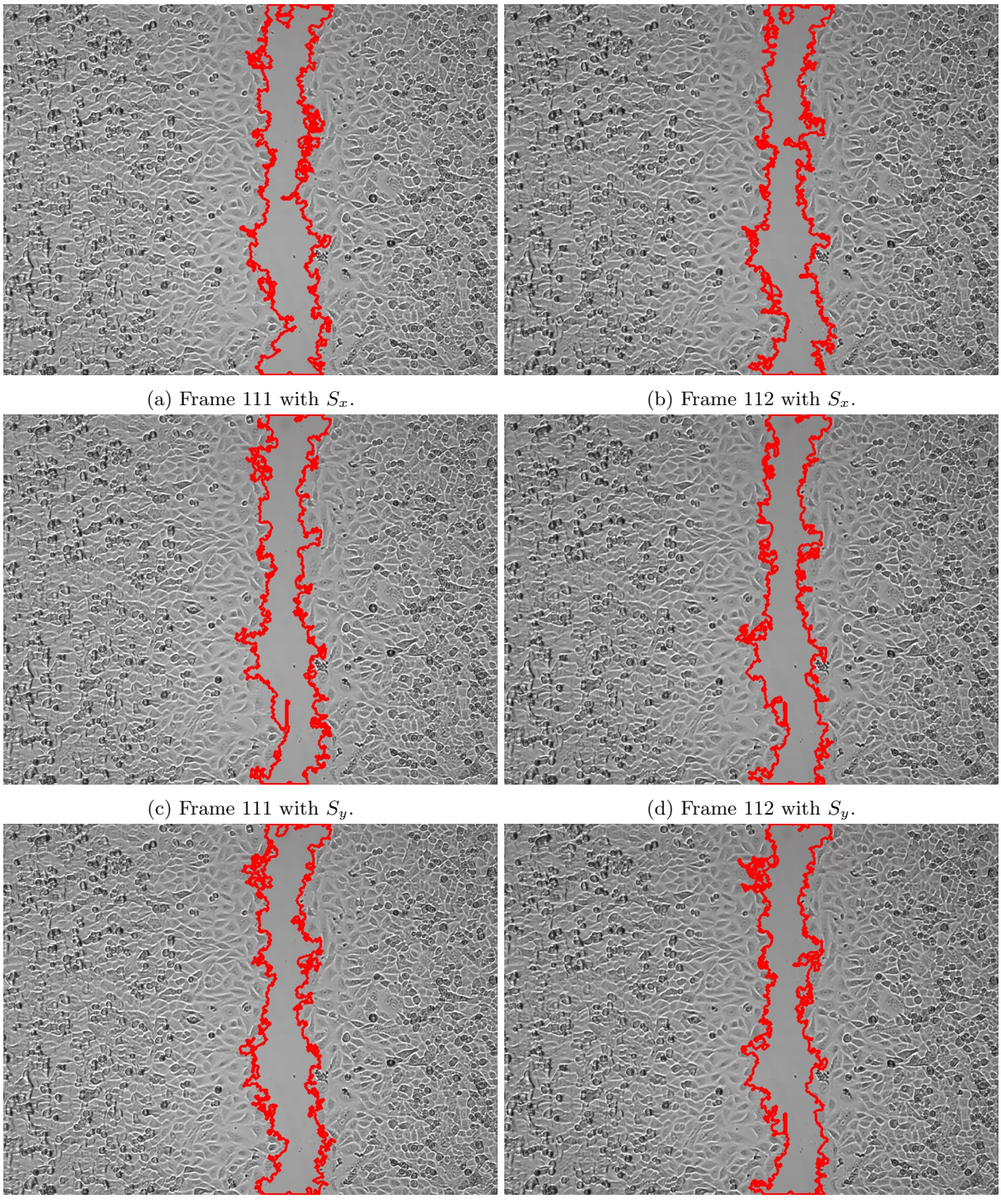
(c) Sobel operation in the y direction.

(d) Combined operation.

Figure 11: Results of the Sobel operator.

In this figure we compared the contours after applying Sobel operation with Figure 6. In Figure 11b we see at the bottom left of the wound a small spike coming out. This is because the Sobel operator in the x direction detected an edge in that location, so that the contour is actually drawn along that edge too. But overall, it can be seen that all the operations could find the edges of the wound, which is a good start.

Now we want to look at how the Sobel operator has performed on the same frames as in Figure 10. Obviously, we do not want to have a sudden overestimation of the area of the wound when the wound did not become bigger as seen in the video. So in the next figures, we will look at these frames and contours.



(a) Frame 111 with S_x.

(b) Frame 112 with S_x.

(c) Frame 111 with S_y.

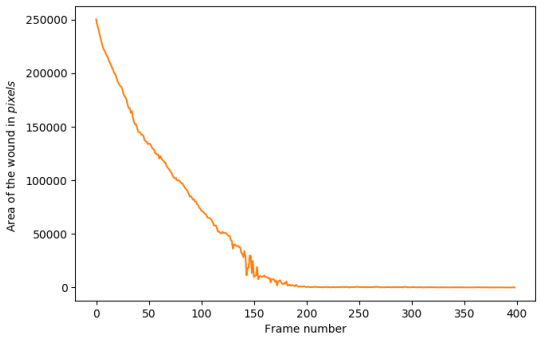
(d) Frame 112 with S_y.

(e) Frame 111 with combined operation.

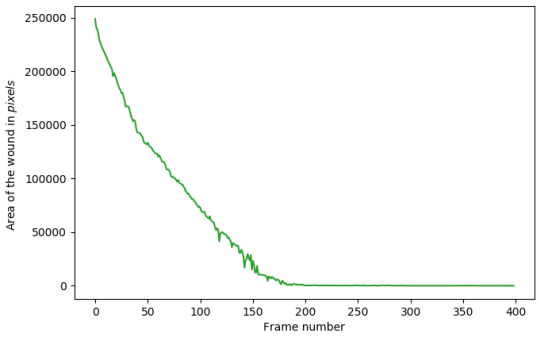
(f) Frame 112 with combined operation.

Figure 12: Results of Sobel operation on the frames 111 and 112.

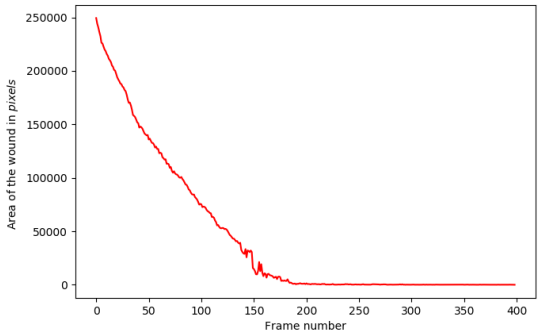
From these figures, we can easily see that the contours are drawn more accurately than the contours in Figure 10. This is because in our previous method, the contour is drawn over a quite large part of the cells just outside of the wound, which gives us an overestimation of the area of the wound. But with the Sobel operator, this is not the case. Although some small loops are still being seen in these contours and some small parts of the cells outside the wound are inside the contour, the estimation of the wound is more accurate when the Sobel operator is applied compared to the images in Figure 10. The results of



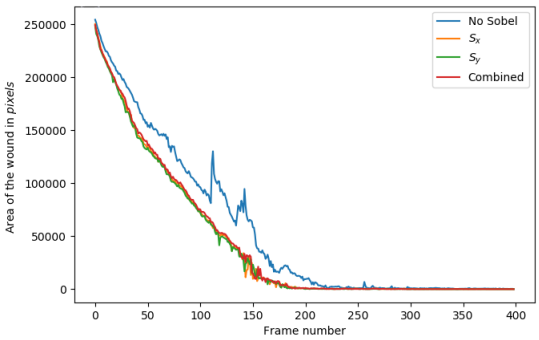
(a) Area of the wound of S_x.



(b) Area of the wound of S_y.



(c) Area of the wound of combined operation.



(d) Areas with and without Sobel operation.

Figure 13: Areas of the wound over time (frames) after Sobel operation.

the area of the wound over time are seen in Figure 13. Now looking at Figure 13d, the Sobel operators do not have the sharp spikes as in the blue area plot, which was the result of our first area estimation. As we would expect, the area of the wound gradually decades over time, without any spontaneous spikes. Moreover, the graphs of S_x, S_y and the combined operations all look very similar to each other, but there is some noise to be seen around frame 150. The reason is because around that frame, the wound (almost) closes in the middle of the wound. Therefore, the wound itself consist of two parts, say an upper and lower part.

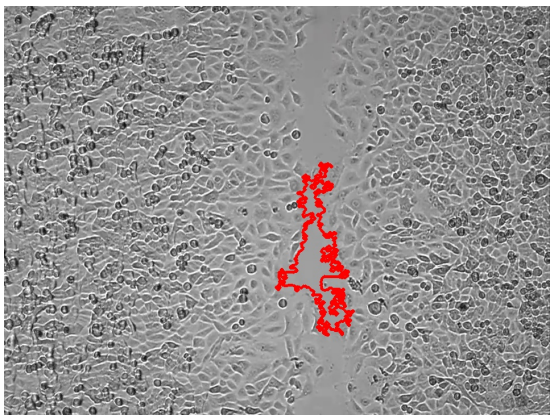


Figure 14: Wound detection only catches a part of the wound when the wound is partly closed in the middle.

Also, because we assumed that the contour that is drawn along the wound is the largest contour of all other contours that may be drawn, we only get to catch one part of the wound instead of both (see Figure 14). This jumping from the upper to lower part of the wound happens for several frames.

Overall, we do not have large spikes anymore after using the Sobel operator, which is certainly an improvement on the first boundary detection method.

4 Cell Detection

Another aspect we want to know from these videos is to find out how many cells are present during the video. Over time, we not only see the cells moving to the wound, some cells split into two or some may die as well. So we want to be able to count the amount of cells (per frame) and moreover to estimate the cell density. The cell density is very useful to know in order to apply one of the existing continuum models:

$$\frac{\partial u}{\partial t} - D\Delta u = f, \tag{4.1}$$

with $u(x, y, t)$ is the cell density, D the diffusion coefficient, f is the source term (cell division rate) and Δ the Laplace operator:

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}.$$

This model is a partial differential equation and is also known as the ***Fisher-Kolmogorov*** model in the context of cellular dynamics. It describes the change in space and time of the concentration of a substance. In our case, the cell density is the substance we want to describe. More of this model will be discussed in Section 5.

As in Section 3.1, our video will be divided into frames, and every frame is going to be analysed. But before we start analysing the frames, let us have a look at some of the properties of each cell. Figure 15

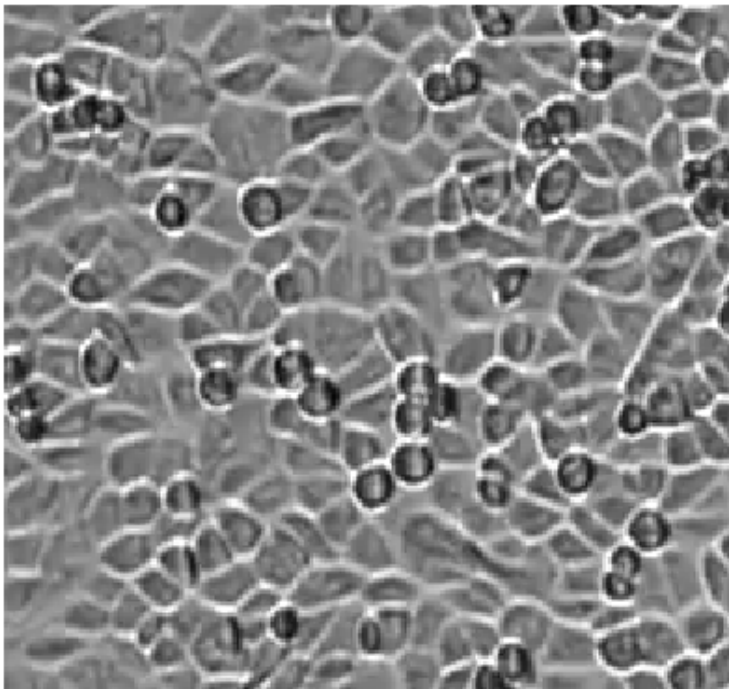


Figure 15: Zoomed image of the cells.

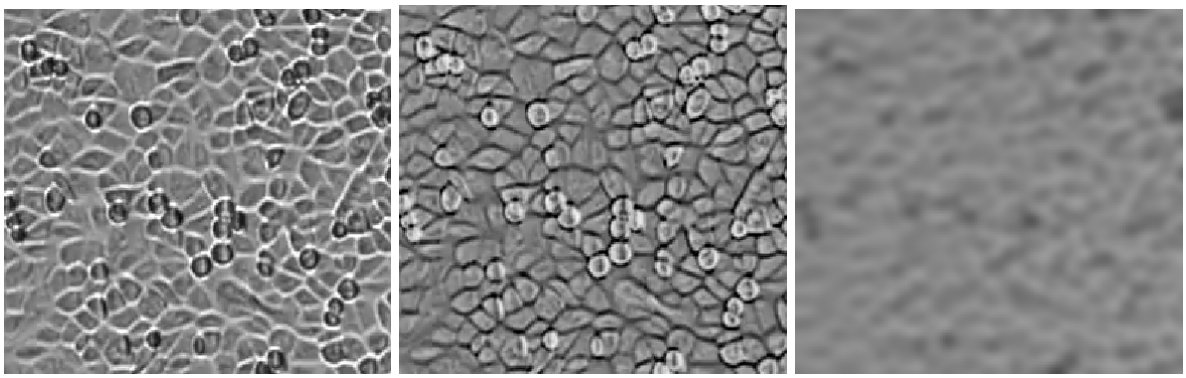
is a zoomed image on the cells of frame 0 with dimensions 243 × 257. If we look at it, then the cells are recognizable because of the following observations:

- Generally, the cells have a clear, bright boundary. This means that the intensity values of the boundary of the cells are very high.
- The interior of the cells are much darker, which means a lower intensity value on the inside. Furthermore, in most of the cells, the intensity values on the inside of the cells are more or less constant.

With these observations, we can start with the detection of the cells.

4.1 Local maxima

The idea of detecting the cells in an image is to find the center of each of the cells (or at least the interior of every cell). This can be done by looking for the local maximum in every cell. We know from our observations that the boundary of every cell is brighter than the interior of the cells. Hence, if we try to find the local maximum in a cell, it would certainly detect the boundary. Thus to avoid this, we are going to invert the image (actually, we could also find the local minima if we did not want to invert the image). By inverting the image, all the dark pixels become bright and the bright pixels become dark. More precisely, if I is an $n \times m$ image, then for every pixel $p(n, m)$ with pixel value $val(n, m)$, the inverted image is the image I_{inv} with pixel values $val_{inv} = 255 - val(n, m)$.



(a) Original image. (b) Inverted image. (c) Blurred image.

Figure 16: Results of inverting and applying a Gaussian blur filter on the image for a suitable σ.

After inverting the image, we have that the interior of the cell has a higher pixel value than the value of the boundary, as seen in Figure 16b. The next step is to apply a Gaussian blur filter on the inverted image. As a result, the image is made smooth and thus the interior of the cells can be detected because the cells will look like 'hills' on the blurred image in Figure 16c.

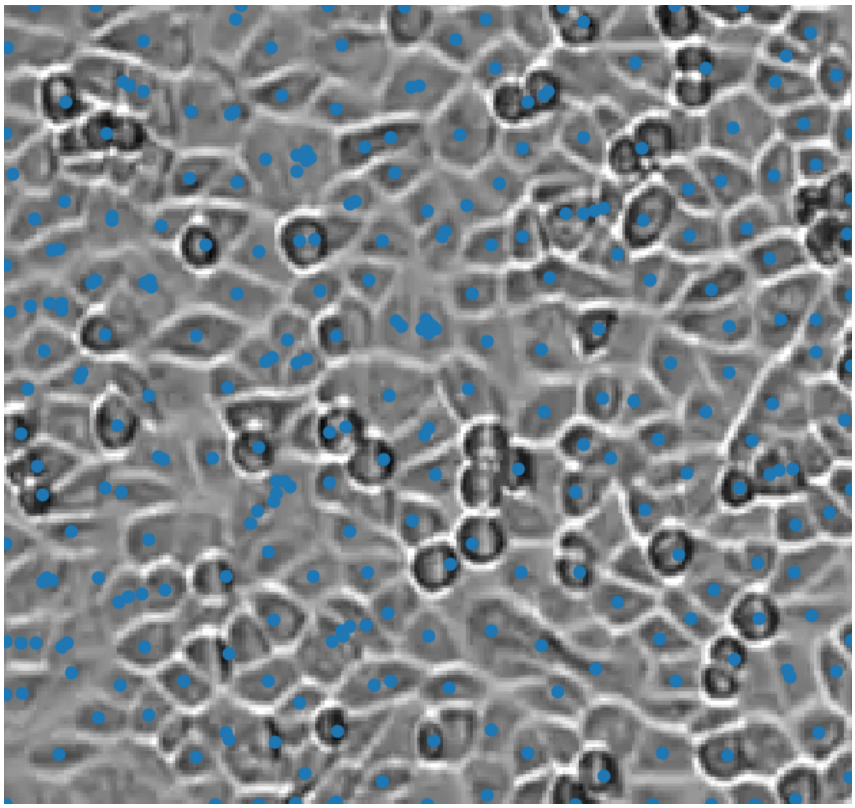
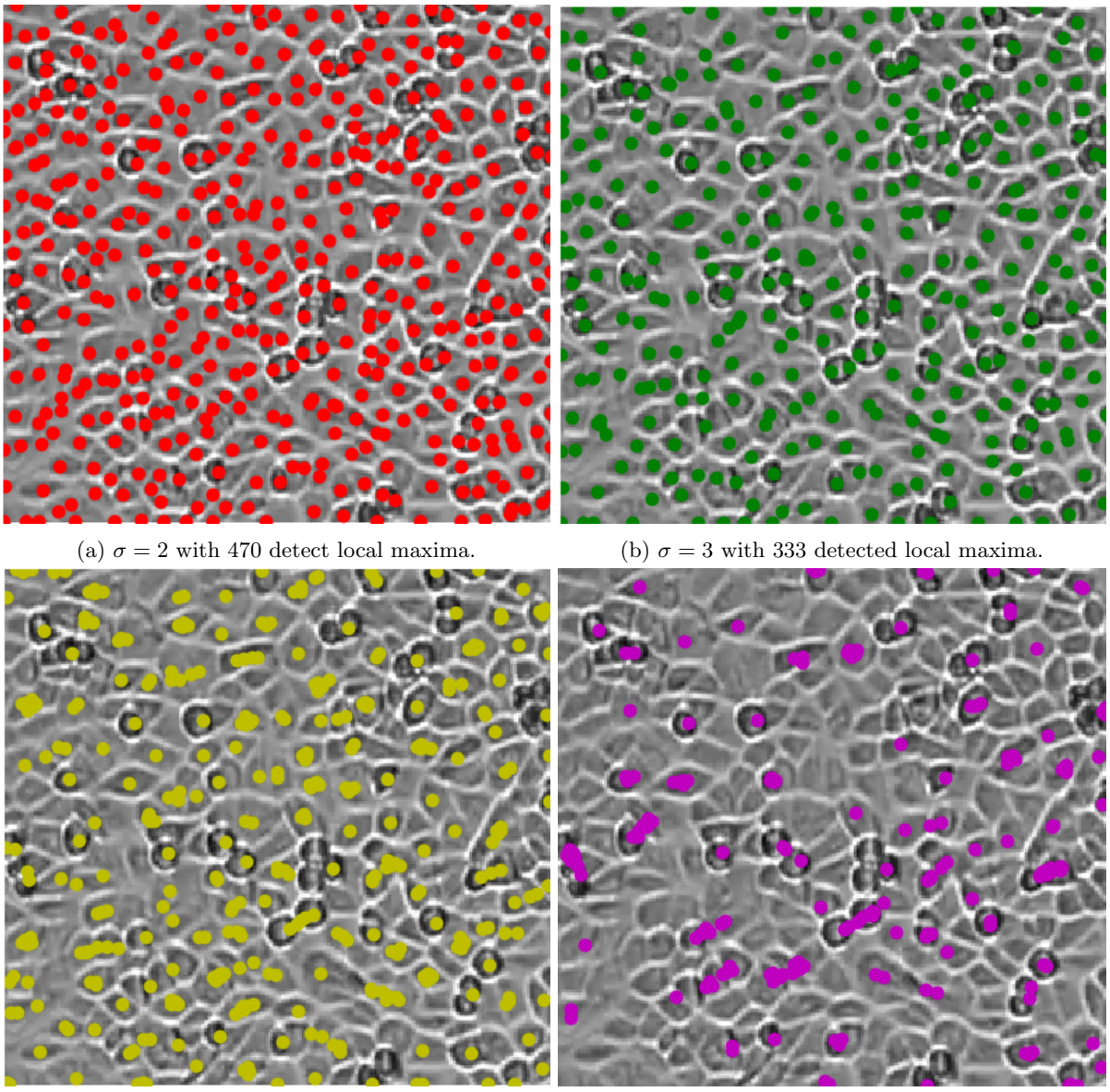


Figure 17: Detection of cells after blurring with $\sigma = 4$.

Now from the library `SciPy`, we can find all the local maxima in the blurred image with the function `filters.maximum_filter` from the package `ndimage`. An example of a result is seen in Figure 17, with $\sigma = 4$. The blue dots in the image represent the local maxima that are found by the `filters.maximum_filter` function. In this image, there are 288 local maxima that have been found. If we count the cells by hand, the number of cells are around 262. We can see that most of the cells are detected, but there are some cells left which were not detected at all. An explanation for this is because the cells which are not detected, do not have clear, bright boundaries compared to their neighbouring cells. Rather, the boundaries look more or less like the interior of the cells. So a group of cells will look like a single cell, and thus not all the local maxima of each cell are detected.

Notice also that some cells have two or more dots very close to each other. The reason for this is that these cells do not have a constant interior, but rather have a 'mountain-like' interior with both low and high intensity values. As a consequence, the function detects multiple local maxima within a single cell.



(a) $\sigma = 2$ with 470 detect local maxima.

(b) $\sigma = 3$ with 333 detected local maxima.

(c) $\sigma = 5$ with 351 detected local maxima.

(d) $\sigma = 6$ with 175 detected local maxima.

Figure 18: Cell center detection for different values of σ's.

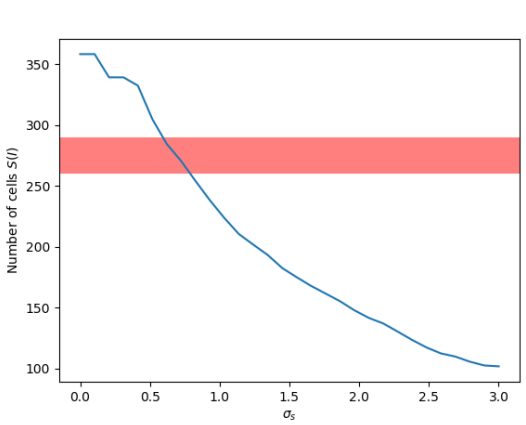
We have also done this for different σ values, see Figure 18. We see that for some values of σ, the total amount of detected cells are less than the actual amount of cells in the image. Also, the dots that have been drawn are more or less clustered in small areas, see for example Figure 18c and Figure 18d. If the σ value is too low, then too many local maxima will be detected, even though all the cells are detected. This means that for many cells in the image, they have multiple dots in their interiors. This can be seen in Figure 18a.

Now the question that may arise is how to choose a suitable σ to detect all (or most) of the cells in an image? To answer it directly, we will make use of the following formula [2]:

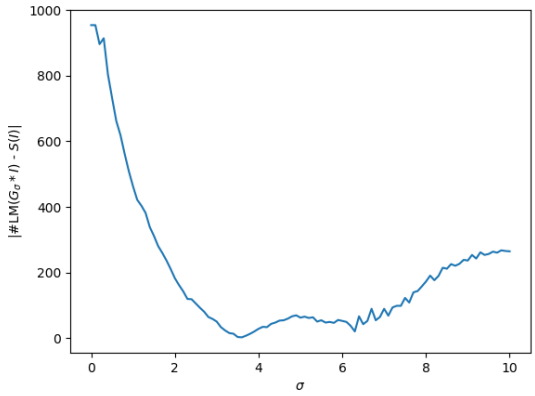
$$\sigma_{opt} = \arg\min_{\sigma>0} |\#\text{LM}(G_\sigma * I) - S(I)|, \tag{4.2}$$

where σ_{opt} is the optimal choice of σ and $\#\text{LM}(G_\sigma * I)$ denotes the number of detected local maxima after blurring the image with a Gaussian filter. $S(I)$ denotes the approximate number of cells calculated using the algorithm described in [2]. This algorithm counts the number of cells in an image (filtered by a Gaussian blur) by counting the number of boundary crossings of the cells in a row and divide that number by 2. Repeat this process for every row, so that the average number of cells in the horizontal direction can be calculated. Then doing the same process for every column, we get the average number of cells in the vertical direction. The total number of cells in an image is then calculated by taking the product of the two averages.

Notice also that we need to choose a suitable σ_s for $S(I)$ if we want to use the algorithm to calculate the number of cells in the image. So we will look at the number of cells $S(I)$ for different values of σ_s. The



(a) Number of cells $S(I)$ for different values of σ_s.



(b) Difference between $\#\text{LM}(G_\sigma * I)$ and $S(I)$ for different values of σ.

Figure 19: Finding the optimal value σ.

red band in Figure 19a denotes the interval [260,290]. Because the number of cells counted by hand was around 262 and our result in Figure 17 detected 288 local maxima, we found that this interval represents a good approximation of the exact number of cells. As we can see, the choice for σ_s for this image lies in the interval $[0.6; 0.8]$. But the σ_s may vary depending on the image that needs to be analysed.

To find σ_{opt}, we will look at different values of σ and find the minimum difference between $\#\text{LM}(G_\sigma * I)$ and $S(I)$. This is visualized in Figure 19b.

From this figure, one can see that the minimum value is achieved if $\sigma_{opt} = 3.6$, where we used $\sigma_s = 0.70$, $S(I) = 273$ and steps of 0.1 to obtain σ_{opt}. But when choosing $\sigma_{opt} = 3.6$, the number of local maxima is actually 338. This number is too high though, compared to when using $\sigma = 4$ with 288 located maxima and 262 when the number is counted by hand. This means that our method is very sensitive to small changes of the parameter σ.

4.2 Histogram equalization

So far, the method we have used in the previous section only consists of the use of a Gaussian blur filter and then finding the local maxima in the image. But as we can see, not all the cells were detected, and some cells were detected more than once. To improve our method, we are going to make use of *Histogram Equalization*. Histogram Equalization is an image processing method used to increase the global image contrast by adjusting the histogram of a given image. When a histogram has many pixels within a (small) range of intensity values, the histogram will be more evenly distributed when the equalization is applied. It spreads out the most frequent intensity values. In this way, areas with a lower local contrast will get a higher contrast.

We will use this technique because the video we have used does not have a high contrast. If we increase the contrast, then we will have a better distinction between the cell's interior and the boundary, and so detecting the cells would be easier.

Histogram equalization goes as follows:

Let I be a given (grayscale) image with dimension $i \times j$ and $V_{i,j}$ the pixel value on pixel (i, j) ranging from 0 to $L-1$. L denotes the number of possible intensity values. Usually, $L = 256$ for grayscale images. Let n_i be the number of pixels that have intensity value k and n the total number of pixels of the image. Then the probability for a pixel to have intensity value i is

$$p_k = \frac{number\ of\ pixels\ with\ intensity\ k}{total\ number\ of\ pixels} = \frac{n_i}{n}, \quad k = 0, 1, \ldots, L-1.$$

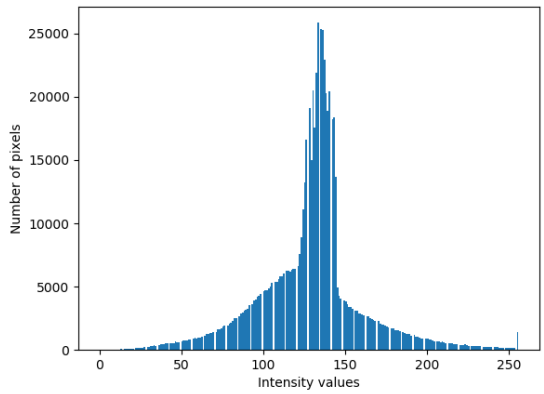
Note that p_k is the image's histogram for pixel value k that is normalized to [0,1].

Now the equalization of the histogram of image I is defined by transforming the intensity value $s \in [0, L-1]$ of I by the function

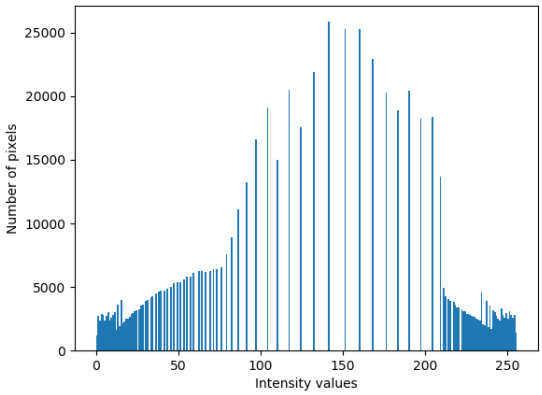
$$T(s) = floor((L-1)\sum_{k=0}^{s} p_k), \tag{4.3}$$

where $floor()$ is the operator that rounds $T(s)$ down to the nearest integer [4],[5].

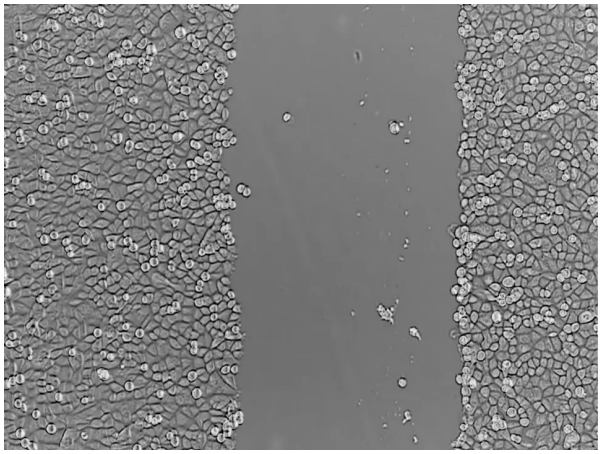
To apply equation (4.3) on the histogram, we used a function in `OpenCV` called `cv2.EqualizeHist` to make this work.



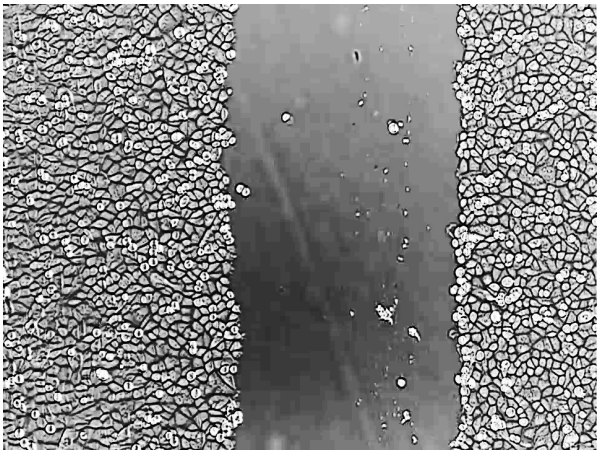
(a) Original histogram of the frame.



(b) Equalized histogram of the frame.



(c) Inverted frame.



(d) Inverted frame after histogram equalization.

Figure 20: Result of histogram equalization.

The result of equalizing a histogram can be seen in Figure 20. Again, we used the histogram of frame 0 to illustrate the effect of histogram equalization. Indeed, the intensity values with a high number of pixels are spread out more evenly over the other values. Having a higher contrast on the image, we see a better distinction between the interior of the cells with a high intensity value and the boundary of the cells with low intensity. Then applying the same method as used in Section 4.1, we can compare these results with our previous method. The result of applying both methods is seen in Figure 21.

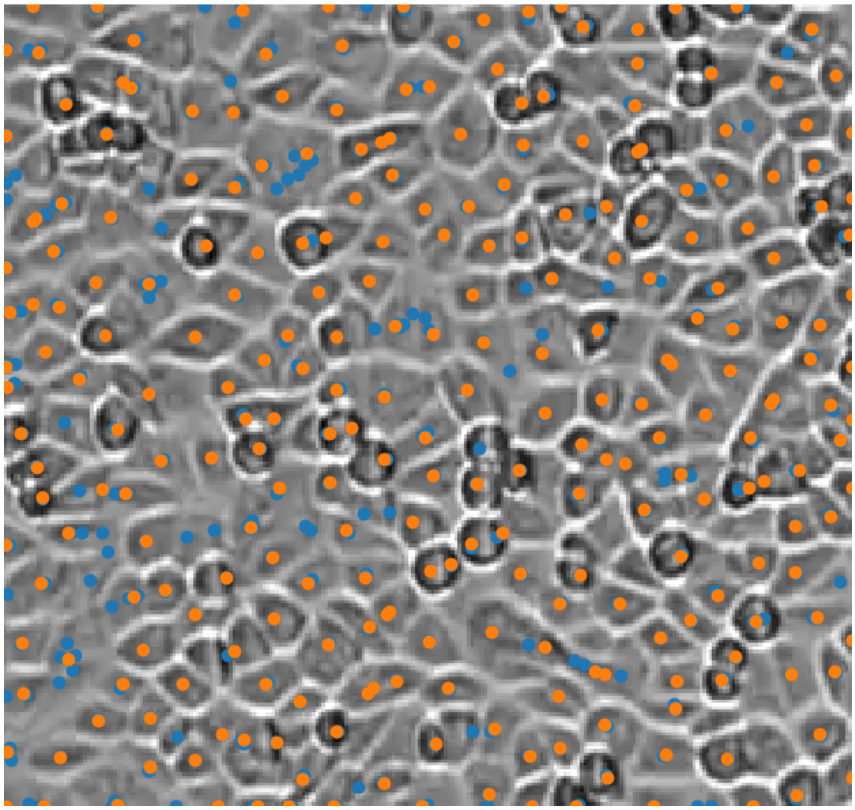


Figure 21: Cell detection with and without histogram equalization (and $\sigma = 3.6$).

The blue dots in this figure represent the detected local maxima without applying histogram equalization and the orange dots are the ones after applying histogram equalization. Here we used $\sigma = 3.6$ because that was the optimal choice for σ we had found. As said earlier, the amount of detected blue dots are 338, whereas for the orange dots there are 269 local maxima detected (the actual cell count is 262 for this frame). Many blue and orange dots overlap each other because they detected the same local maxima. Also, we plotted the blue dots first and then orange. So at first sight it looks like we have more orange dots than blue, but this is not the case.

As we can see, almost all cells are detected after histogram equalization as well, but some cells still have multiple dots in their interior. However, we see that there are less clustered orange dots compared to blue, which explains why there are less local maxima detected after equalization. Moreover, the number of orange dots comes closer to what we have counted by hand and what we have calculated with $S(I)$ than the number of blue dots. Also this number is within the 'acceptable' range of $[260, 290]$.

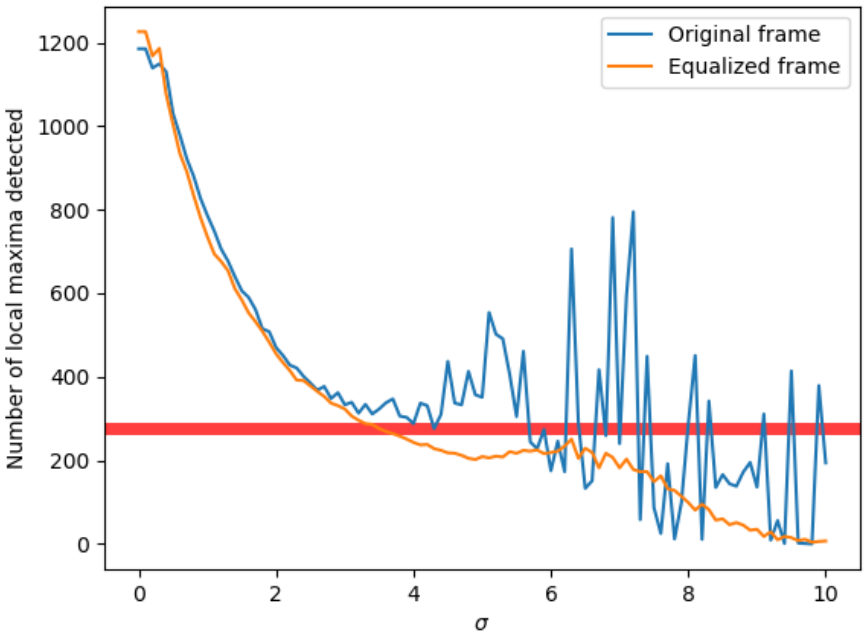


Figure 22: Number of local maxima detection for different values of σ.

Figure 22 shows us the number of local maxima detection for $\sigma \in [0, 10]$ with steps 0.1 and the red band represents the acceptable range of $[260, 290]$. What stands out the most is that the detection of cells swings heavily when no equalization is applied, whereas the detection is far more stable when equalization has been used. Following the blue line, we see that for some values of σ the number of local maxima is indeed within the red band. But as mentioned before under Figure 18, the maxima are more or less clustered in small areas. So even though the number of local maxima is within the red band, not every cell will be detected. Thus from these observations, it seems like using histogram equalization before detecting the local maxima in an image will give us a better approximation of the number of cells in an image.

5 Continuum model

In this section, we will finally apply the continuum model we have mentioned before in equation (4.1):

$$\frac{\partial u}{\partial t} - D\Delta u = f. \tag{5.1}$$

To solve this equation, we are going to discretize it in space and time. First, we set up a grid of points over the domain D of the image I, like so in Figure 23. The black box represents the frame of the image,

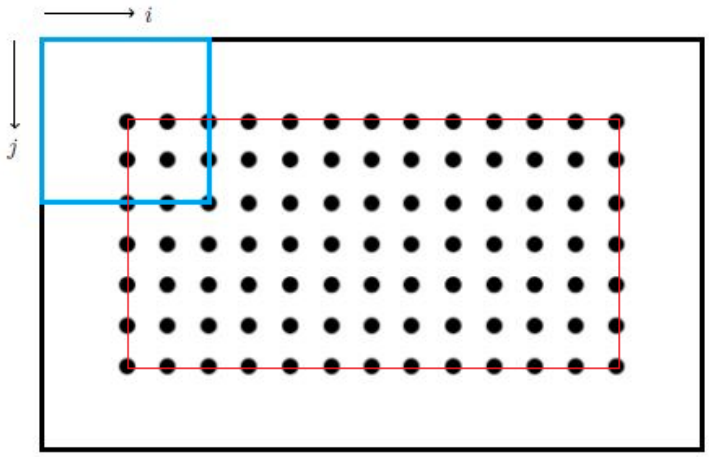


Figure 23: Grid of points

with length L and width W. The red line with the corresponding points is the boundary ∂D and the distance between the grid points is h. The blue square represents the small window with length a where the amount of cells will be counted in that small window. Then the grid point (x_i, y_j) is the center of the blue square, with value $u(x_i, y_j) = u_{i,j}$[4]. Here we have $x_i = a/2 + ih$, for $i \in \{0, 1, \dots, M\}$, where $W - a = Mh$ (otherwise the blue square exceeds the frame). Same for $y_j = a/2 + jh$, for $j \in \{0, 1, \dots, N\}$, where $L - a = Nh$. The video we analysed so far had $L = 720$ pixels and $W = 960$, where we chose $h = 10$ and $a = 80$. Thus $M = 88$ and $N = 64$ in our case. In the next iterations, the blue square moves lexicographically, which means it runs through all x_i for y_0, then it runs through all x_i for y_1 etc.

The Laplace operator consists of two partial derivatives. We can use central differences to approximate these derivatives in each coordinate (x_i, y_j):

$$(\Delta u)_{i,j} = \frac{\partial^2 u_{i,j}}{\partial x^2} + \frac{\partial^2 u_{i,j}}{\partial y^2} \approx \frac{u_{i-1,j} - 2u_{i,j} - u_{i+1,j}}{h^2} + \frac{u_{i,j-1} - 2u_{i,j} - u_{i,j+1}}{h^2}, \tag{5.2}$$

with $i \in \{1, 2, \dots, M-1\}$ and $j \in \{1, 2, \dots N-1\}$. Combining the two derivatives gives us

$$\frac{\partial^2 u_{i,j}}{\partial x^2} + \frac{\partial^2 u_{i,j}}{\partial y^2} \approx \frac{u_{i-1,j} - 4u_{i,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}}{h^2}. \tag{5.3}$$

The boundary values are denoted as $u_{:,0}$, $u_{M,:}$, $u_{0,:}$ and $u_{:,N}$, where $u_{:,0}$ means $u_{i,0}$ for $i = 0, ..M$. The solution vector $\hat{\mathbf{u}} \in \mathbb{R}^{(M-1)(N-1)}$ (without the boundary values) would be

$$\hat{\mathbf{u}} = (u_{1,1}, u_{2,1}, \dots, u_{M-1,1}, u_{1,2}, u_{2,2}, \dots u_{M-1,2}, \dots, u_{1,N-1}, u_{2,N-1}, \dots, u_{M-1,N-1})^\top$$

[4]It must actually be written as $u_{i,j}^{(k)}$, with k the frame number of the video, because the $u_{i,j}^k$ changes over time. The same holds for f. But for now, we omit the notation for convenience.

Now define the $(M-1)\times(M-1)$ matrix

$$Q = \begin{bmatrix} -4 & 1 & & & & \\ 1 & -4 & 1 & & & \\ & 1 & -4 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -4 & 1 \\ & & & & 1 & -4 \end{bmatrix} \tag{5.4}$$

and I the $(M-1)\times(M-1)$ identity matrix. Because our solution vector $\hat{\mathbf{u}} \in \mathbb{R}^{(M-1)(N-1)}$, we will get a linear system of $(M-1)(N-1)\times(M-1)(N-1)$:

$$\Delta u \approx A\hat{\mathbf{u}} + \mathbf{b}$$

, with A the $(M-1)(N-1)\times(M-1)(N-1)$ block matrix defined by:

$$A = \begin{bmatrix} Q & I & & & & \\ I & Q & I & & & \\ & I & Q & I & & \\ & & \ddots & \ddots & \ddots & \\ & & & I & Q & I \\ & & & & I & Q \end{bmatrix}, \tag{5.5}$$

and $\mathbf{b}$ the boundary values defined by:

$$\mathbf{b} = (u_{1,0}, u_{2,0}, \ldots, u_{M-1,0}, 0, \ldots, 0, u_{1,N-1}, \ldots, u_{M-1,N-1})^\top +$$
$$(u_{0,1}, 0, \ldots, 0, u_{M,1}, u_{0,2}, 0, \ldots, 0, u_{M,2}, \ldots, \ldots, u_{0,N-1}, 0, \ldots, 0, u_{M,N-2}, u_{0,N-1}, 0, \ldots, 0, u_{M,N-1})^\top,$$

with the first term of $\mathbf{b}$ the 'north-south'-boundary and the second term the 'west-east'-boundary. These are Dirichlet boundary conditions, since we can determine the boundary values explicitly.

Now using finite difference for the time derivative, we get:

$$\frac{\partial u}{\partial t} \approx \frac{\hat{\mathbf{u}}^{(k+1)} - \hat{\mathbf{u}}^{(k)}}{\Delta t}, \quad k = 0, 1, 2, \ldots \tag{5.6}$$

Combining all the results from above, we obtain the finite difference scheme using the Forward Euler method:

$$\hat{\mathbf{u}}^{(k+1)} = \mathbf{u}^{(k)} + \frac{\Delta t D_{(k)}}{h^2}(A\mathbf{u}^{(k)} + \mathbf{b}^{(k)}) + \Delta t \mathbf{f}^{(k)} \tag{5.7}$$

We use the Forward Euler method because it is a relatively fast algorithm and it is easy to implement. In this case, Δt is set to 0.1. This is because the video lasts for 39 seconds, whereas the video has 399 frames, so that the time between two frames is about 0.1 seconds.

5.1 Finding the parameter D

Now we want to find a suitable parameter $D_{(k)}$ such that the difference between the real data $\mathbf{u}^{(k+1)}$ and the finite difference scheme $\hat{\mathbf{u}}^{(k+1)}$ is minimized:

$$\|\mathbf{u}^{(k+1)} - \hat{\mathbf{u}}^{(k+1)}\|_2^2. \tag{5.8}$$

For convenience, we write down $\hat{\mathbf{u}}^{(k+1)} = \mathbf{u}^{(k)} + D_{(k)}B\mathbf{u}^{(k)} + D_{(k)}\mathbf{v}^{(k)} + \mathbf{g}^{(k)}$, with $B = \frac{\Delta t}{h^2}A$ and $\mathbf{v}^{(k)} = \frac{\Delta t}{h^2}\mathbf{b}^{(k)}$ and $\mathbf{g}^{(k)} = \Delta t \mathbf{f}^{(k)}$. But first, we need to write out the term (carefully) before we can find

the suitable parameter. This goes as follows:

$$\|\mathbf{u}^{(k+1)} - \hat{\mathbf{u}}^{(k+1)}\|_2^2 = \|\mathbf{u}^{(k+1)} - (\mathbf{u}^{(k)} + D_{(k)}B\mathbf{u}^{(k)} + D_{(k)}\mathbf{v}^{(k)} + \mathbf{g}^{(k)}\|_2^2 \tag{5.9}$$

$$= \|\mathbf{u}^{(k+1)} - ([I + D_{(k)}B]\mathbf{u}^{(k)} + D_{(k)}\mathbf{v}^{(k)} + \mathbf{g}^{(k)})\|_2^2. \tag{5.10}$$

This is an inner product, written as

$$\langle \mathbf{u}^{(k+1)} - [I + D_{(k)}B]\mathbf{u}^{(k)} - D_{(k)}\mathbf{v}^{(k)} - \mathbf{g}^{(k)}, \mathbf{u}^{(k+1)} - [I + D_{(k)}B]\mathbf{u}^{(k)} - D_{(k)}\mathbf{v}^{(k)} - \mathbf{g}^{(k)} \rangle. \tag{5.11}$$

Now equation 5.11 can be written in the form:

$$\|\mathbf{u}^{(k+1)} - \hat{\mathbf{u}}^{(k+1)}\|_2^2 = a_k D_{(k)}^2 + 2b_k D_{(k)} + c_k, \tag{5.12}$$

with the coefficients

$$a_k = \langle \mathbf{v}^{(k)}, \mathbf{v}^{(k)} \rangle + \langle B\mathbf{u}^{(k)}, B\mathbf{u}^{(k)} \rangle + 2\langle \mathbf{v}^{(k)}, B\mathbf{u}^{(k)} \rangle \tag{5.13}$$

$$b_k = -\langle B\mathbf{u}^{(k)}, \mathbf{u}^{(k+1)} \rangle - \langle \mathbf{v}^{(k)}, \mathbf{u}^{(k+1)} \rangle + \langle B\mathbf{u}^{(k)}, \mathbf{u}^{(k)} \rangle + \langle B\mathbf{u}^{(k)}, \mathbf{g}^{(k)} \rangle + \langle \mathbf{u}^{(k)}, \mathbf{v}^{(k)} \rangle + \langle \mathbf{v}^{(k)}, \mathbf{g}^{(k)} \rangle \tag{5.14}$$

$$c_k = -2\langle \mathbf{u}^{(k)}, \mathbf{u}^{(k+1)} \rangle - 2\langle \mathbf{u}^{(k+1)}, \mathbf{g}^{(k)} \rangle + 2\langle \mathbf{u}^{(k)}, \mathbf{g}^{(k)} \rangle + \langle \mathbf{u}^{(k)}, \mathbf{u}^{(k)} \rangle, + \langle \mathbf{g}^{(k)}, \mathbf{g}^{(k)} \rangle + \langle \mathbf{u}^{(k+1)}, \mathbf{u}^{(k+1)} \rangle \tag{5.15}$$

Taking the derivative in $D_{(k)}$ and setting the equation to zero gives us an extremum (5.8):

$$D_{(k)} = -\frac{b_k}{a_k}. \tag{5.16}$$

Now, we do not know our source term $\mathbf{g}^{(k)}$, so we set it to zero instead to be able to calculate the $D_{(k)}$'s. The graph of D_k can be seen in Figure 24.

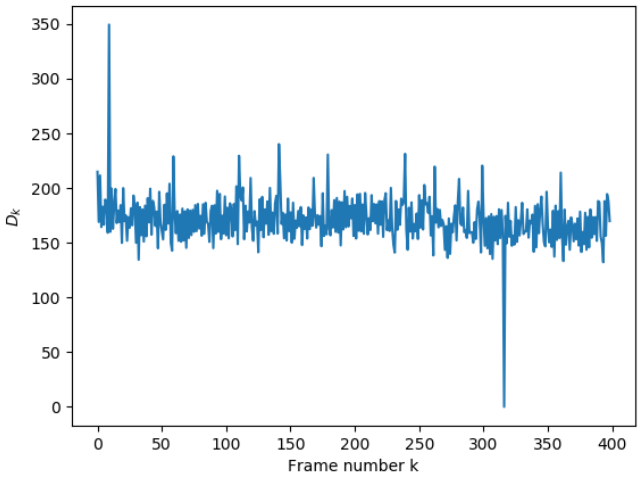


Figure 24: Plot of the D_k's for every frame number k.

We see that the parameter $D_{(k)}$ differs for every frame k, with an interesting spike at frame 316. Apparently, the value there is zero. Now we want to look for a single constant D such that the overall difference between the measured data and the model is minimal. Therefore, we apply the method of least-squares, which is a method widely used in statistics. In our case, the method of least squares becomes:

$$\mathbf{F}(D) = \sum_{k=1}^{m} \|\mathbf{u}^{(k+1)} - \hat{\mathbf{u}}^{(k+1)}\|_2^2, \tag{5.17}$$

starting with $k = 0$, $\mathbf{u}^{(0)} = \hat{\mathbf{u}}^{(0)}$.

To find the parameter D that fits the model to the real data, we need to take the derivative of (5.17) and then set it to zero:

$$\mathbf{F}(D) = \sum_{k=1}^{m} \|\mathbf{u}^{(k+1)} - \hat{\mathbf{u}}^{(k+1)}\|_2^2 \tag{5.18}$$

$$= \sum_{k=1}^{m} \left(a_k D^2 + 2b_k D + c_k\right) \tag{5.19}$$

$$\mathbf{F}'(D) = \sum_{k=1}^{m} (2a_k D + 2b_k) \tag{5.20}$$

$$= D\left(2\sum_{k=1}^{m} a_k\right) + \sum_{k=1}^{m} 2b_k = 0. \tag{5.21}$$

$$\rightarrow D_{LS} = -\frac{\sum_{k=1}^{m} b_k}{\sum_{k=1}^{m} a_k}. \tag{5.22}$$

So here we found the optimal parameter D_{LS} in least-squares. Using the formula for D, we obtain the value $D_{LS} \approx 170.27449884853527$. We can also look at the average of the $D_k's$, with

$$D_{avg} = \frac{1}{m}\sum_{k=1}^{m} D_k = \frac{1}{m}\sum_{k=1}^{m}\left(-\frac{b_k}{a_k}\right). \tag{5.23}$$

Then from Figure 24, the average $D_{avg} \approx 170.98490044714137$. Note that the dimensions of D_{LS} and D_{avg} are in pixels and seconds, whereas the actual units are in μm and hours.

5.2 Results of modelling

In this section, we are going to show the results of modelling with the parameters D_{LS} and D_{avg}. Remember that we do not know our source term, so we set it to zero. In this way, we can see if diffusion will take place in our model.

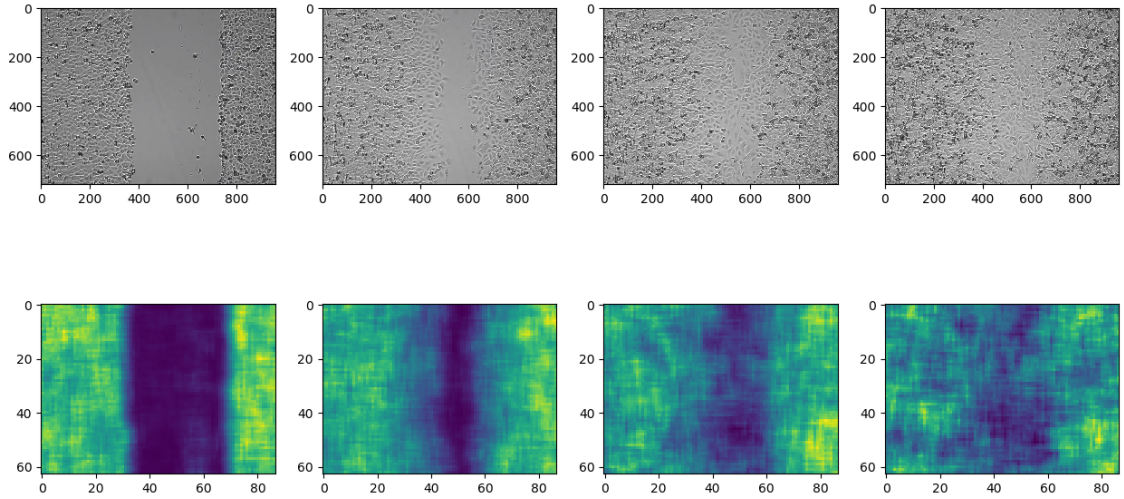


Figure 25: Real measurements of several frames and their corresponding cell densities. From left to right, we have frame 0, 100, 200 and 300.

Figure 25 shows us the real measurements of the cell density, with low cell density at the wound and higher density outside the wound, as we would expect. Now we want to be able to have similar results when we model this process using the finite difference scheme.

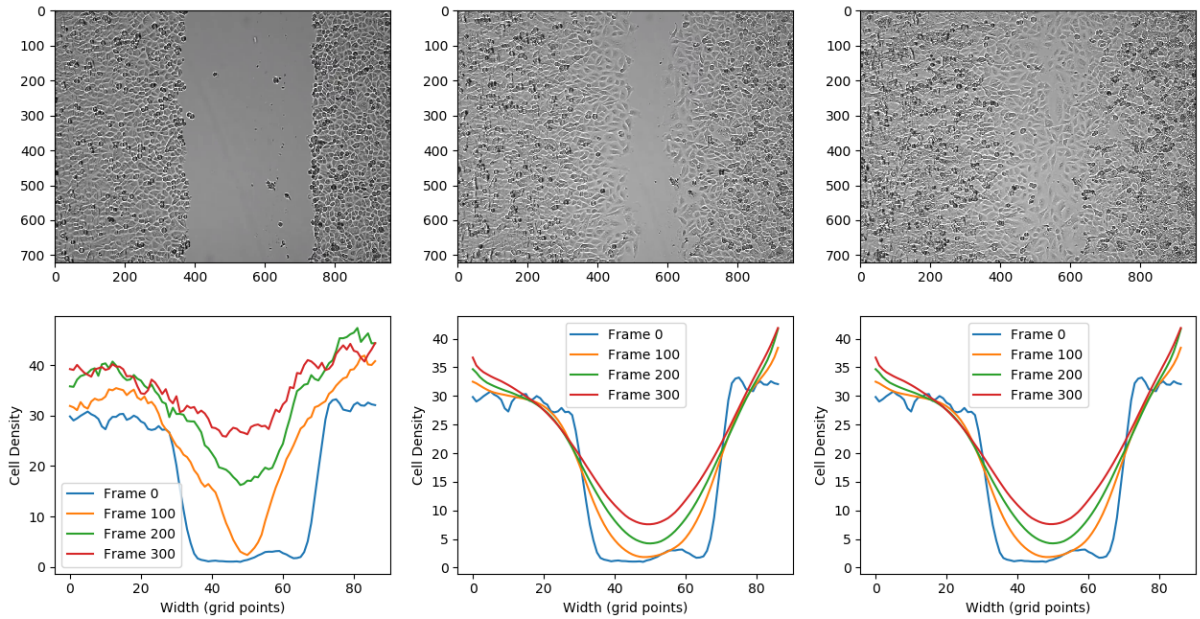


Figure 26: Comparison of the results of modelling with D_{LS} and D_{avg} and the real measurements. From left to right, the upper frames are frame 0, frame 100 and frame 200. The lower figures are the average cell density plots in vertical direction for frames 0, 100, 200 and 300. From left to right, we have the real measurements, then the plot from the model with D_{LS} and right the plot from the model with D_{avg}.

Figure 26 shows us the results of the real data and the data from the model. We took the average cell density in vertical direction for the frames, so that we can see how that changes over time. The actual data shows us that the average cell density increases over time, especially in the wound, so diffusion does take place here. Also, the cell density in the surrounding tissue increases, which indicates that cells have been dividing during the process as well.

In the plots with parameter D_{LS} and D_{avg} we see similar results compared to the real measurements. Both have increase in density in the wound and also in the boundary. Because the difference between the values of D_{LS} and D_{avg} is small, the plots of those two look almost exactly the same.

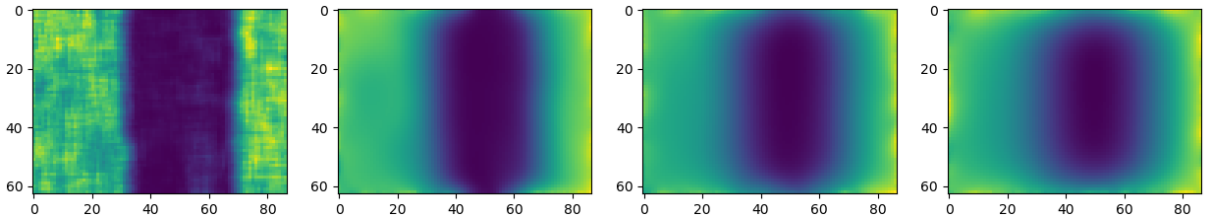


Figure 27: Cell density plot with parameter $D_{LS} \approx 170.27449884853527$. From left to right, the corresponding frames are 0, 100, 200 and 300.

Figure 27 shows us the cell density 'maps' of frame 0, 100, 200 and 300, with parameter D_{LS}. The images look smooth, so that we see that diffusion takes place in our model, which is what we wanted.

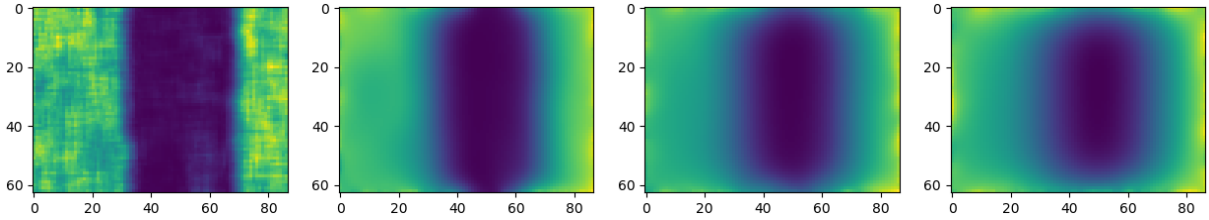


Figure 28: Cell density plots with parameter $D_{avg} \approx 170.98490044714137$. From left to right, the corresponding frames are 0, 100, 200 and 300.

Figure 28 are the results with $D_{avg} \approx 170.98490044714137$. It should come as no surprise that the images in Figure 27 and Figure 28 look exactly the same. Of course the images with parameters D_{LS} and D_{avg} are not fully matched to the real measurements, because we did not consider the cell divisions that took place during the video as well. However, the method we used to derive a suitable parameter gave us a good estimation of the parameter, where diffusion can be seen which is similar to the real measurements.

5.3 Source term

Although we did not model the diffusion of cells with a source term, the method to find a suitable parameter (or parameters) described in Section 5.1 can be used as well with a source term. For example, let f in equation (5.1) be the *mitotic generation* term, given by:

$$f = \alpha u \left(1 - \frac{u}{u_0}\right), \tag{5.24}$$

where α is a positive parameter and u_0 the unwounded cell density. This term describes the cell growth in a logistic form, which is mentioned in [9]. Then the diffusion equation becomes:

$$\frac{\partial u}{\partial t} - D\Delta u = \alpha u \left(1 - \frac{u}{u_0}\right). \tag{5.25}$$

We will not go too deep into detail about this source term, but we will rather focus on how to find the parameters D and α.

From equation (5.11), we have written out the difference between the measured data and the modelled data, with a general source term $\mathbf{g}^{(k)}$. Now , if we substitute the source term in equation (5.11), then the equation can be written as:

$$\|\mathbf{u}^{(k+1)} - \hat{\mathbf{u}}^{(k+1)}\|_2^2 = a_k D_{(k)}^2 + 2D_{(k)} b_k + 2D_{(k)} \alpha c_k + 2\alpha d_k + \alpha^2 e_k + f_k, \tag{5.26}$$

with the coefficients:

$$a_k = \langle \mathbf{v}^{(k)}, \mathbf{v}^{(k)} \rangle + \langle B\mathbf{u}^{(k)}, B\mathbf{u}^{(k)} \rangle + 2\langle \mathbf{v}^{(k)}, B\mathbf{u}^{(k)} \rangle \tag{5.27}$$

$$b_k = -\langle B\mathbf{u}^{(k)}, \mathbf{u}^{(k+1)} \rangle - \langle \mathbf{v}^{(k)}, \mathbf{u}^{(k+1)} \rangle + \langle B\mathbf{u}^{(k)}, \mathbf{u}^{(k)} \rangle + \langle \mathbf{u}^{(k)}, \mathbf{v}^{(k)} \rangle \tag{5.28}$$

$$c_k = \langle B\mathbf{u}^{(k)}, \mathbf{u}^{(k)} \circ (\mathbf{1} - \frac{\mathbf{u}^{(k)}}{u_0}) \rangle + \langle \mathbf{v}^{(k)}, \mathbf{u}^{(k)} \circ (\mathbf{1} - \frac{\mathbf{u}^{(k)}}{u_0}) \rangle \tag{5.29}$$

$$d_k = \langle \mathbf{u}^{(k)}, \mathbf{u}^{(k)} \circ (\mathbf{1} - \frac{\mathbf{u}^{(k)}}{u_0}) \rangle - \langle \mathbf{u}^{(k+1)}, \mathbf{u}^{(k)} \circ (\mathbf{1} - \frac{\mathbf{u}^{(k)}}{u_0}) \rangle \tag{5.30}$$

$$e_k = \langle \mathbf{u}^{(k)} \circ (\mathbf{1} - \frac{\mathbf{u}^{(k)}}{u_0}), \mathbf{u}^{(k)} \circ (\mathbf{1} - \frac{\mathbf{u}^{(k)}}{u_0}) \rangle \tag{5.31}$$

$$f_k = \langle \mathbf{u}^{(k)}, \mathbf{u}^{(k)} \rangle + \langle \mathbf{u}^{(k+1)}, \mathbf{u}^{(k+1)} \rangle - 2\langle \mathbf{u}^{(k)}, \mathbf{u}^{(k+1)} \rangle. \tag{5.32}$$

Here, $\mathbf{u}^{(k)} \circ (\mathbf{1} - \frac{\mathbf{u}^{(k)}}{u_0})$ is discretized, with $\circ$ the Hadamard product and $\mathbf{1}$ a column vector with all the elements having value 1. So our source term is still a vector.

Now, define the functional

$$\mathbf{F}(D,\alpha) = \sum_{k=1}^{m} \|\mathbf{u}^{(k+1)} - \hat{\mathbf{u}}^{(k+1)}\|_2^2 \tag{5.33}$$

$$= \sum_{k=1}^{m} \left(a_k D^2 + 2Db_k + 2D\alpha c_k + 2\alpha d_k + \alpha^2 e_k + f_k\right). \tag{5.34}$$

Then the optimal parameters D and α are found by taking the gradient of (5.33) and setting the partial derivatives to zero:

$$\begin{cases} \dfrac{\partial \mathbf{F}}{\partial D} = 0 \rightarrow \left(\displaystyle\sum_{k=1}^{m} a_k\right) D + \left(\sum_{k=1}^{m} c_k\right) \alpha = -\sum_{k=1}^{m} b_k, & (5.35) \\ \dfrac{\partial \mathbf{F}}{\partial \alpha} = 0 \rightarrow \left(\displaystyle\sum_{k=1}^{m} c_k\right) D + \left(\sum_{k=1}^{m} e_k\right) \alpha = -\sum_{k=1}^{m} d_k. & (5.36) \end{cases}$$

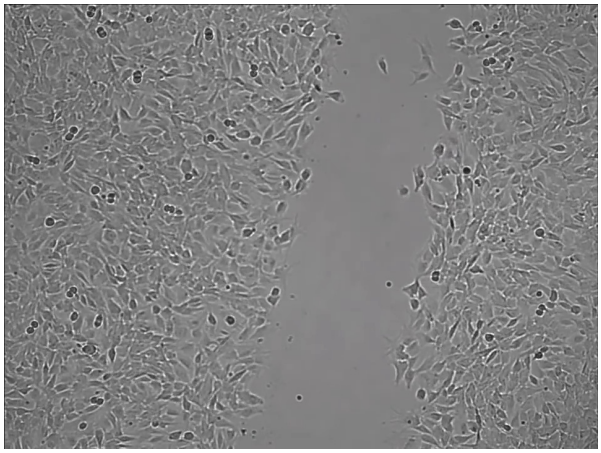
This is a system of equations which can be solved for D and α, which will give us the optimal values for the parameters in least-squares.

6 Results from other videos

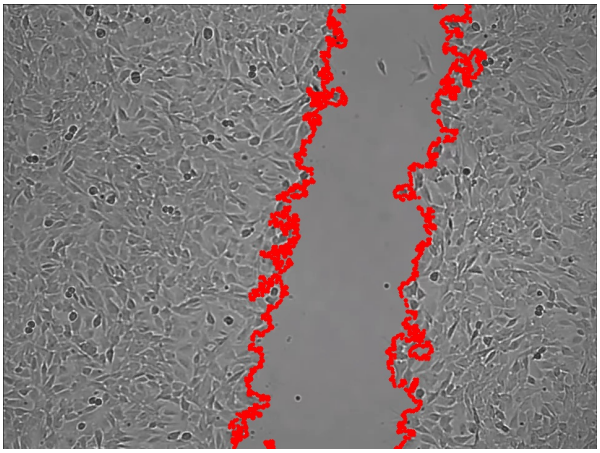
In this section, we will be looking at two other videos to see how our methods work on these videos as well. We will be looking at the area calculations and also the cell detection. The second video[5] we are going to analyse looks similar to the first one, so we will see in the next figures how it works out.

6.1 Boundary detection

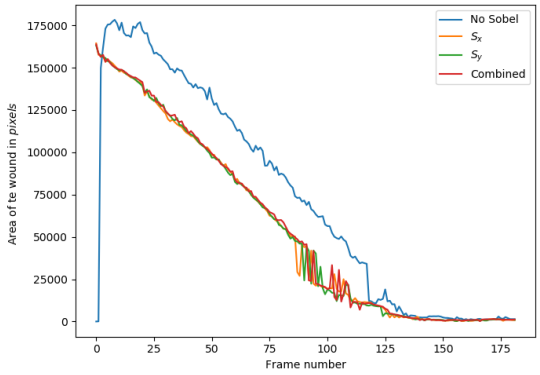
First we will look at the area calculation of the wound, using the Sobel filter and without the Sobel filter in Figure 29. Obviously, the blue line gives an odd result compared to the Sobel filter lines. The



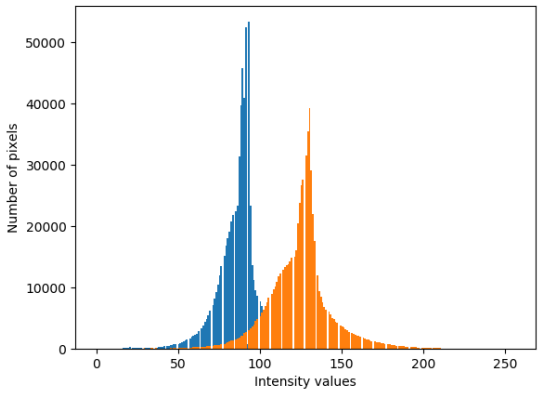
(a) A frame from the video.



(b) Contour of the wound.



(c) Area calculation of the wound.



(d) Blue histogram at the beginning of the video and orange the histogram after the beginning.

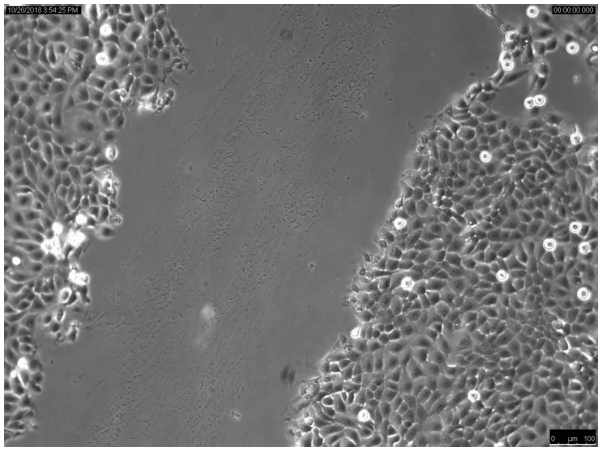
Figure 29: Results of area calculation of the wound with provided with two examples of histograms of the video.

blue line starts with zero area of the wound and then suddenly it makes a huge jump. The reason for this happening is because the video is darker at the beginning compared to the rest of the video (Figure 29d). So the intensity values are lower at the beginning, and the method without the Sobel filter depends heavily on the intensity of the video, by putting thresholds. The thresholds values will be in the range of the orange histogram, causing the method to not detect the wound at the beginning.

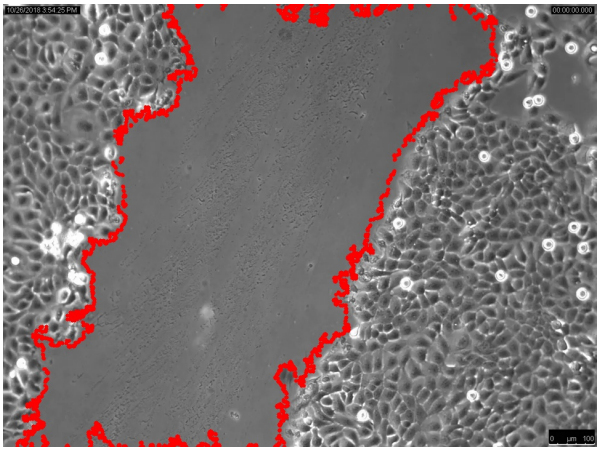
[5] https://www.youtube.com/watch?v=O8CQSva1G50

Another thing that stands out is that every line in Figure 29c has some jumps around frame 100 for the Sobel filters and around frame 120 without the Sobel filter. The reason for this is the same as back then in Figure 13a. The wound actually closes faster in the middle of the wound, causing the wound to have an upper part and lower part, making it difficult to catch both of the parts. But overall, the graphs look similar to the results in Figure 13d.

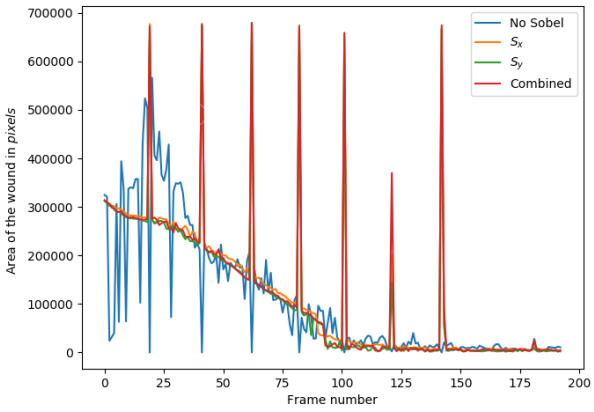
We will also calculate the area of the wound in the third video[6]. This video is different than the previous videos, so that we can see how well the methods perform under different circumstances. The results in



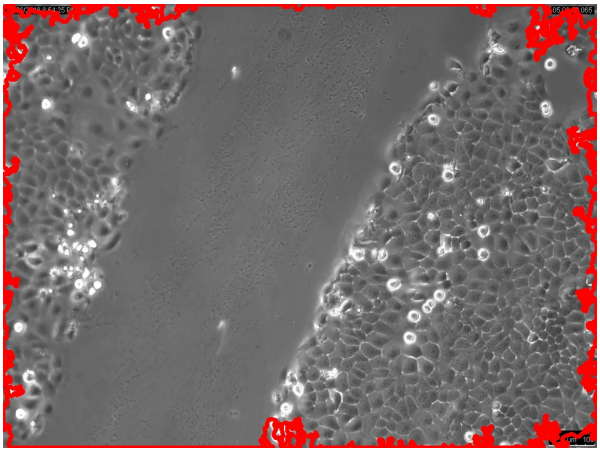
(a) Frame of the third video.



(b) Contour of the wound.



(c) Area calculation of the wound.



(d) Contour at frame 19, which explains the spikes in the graphs.

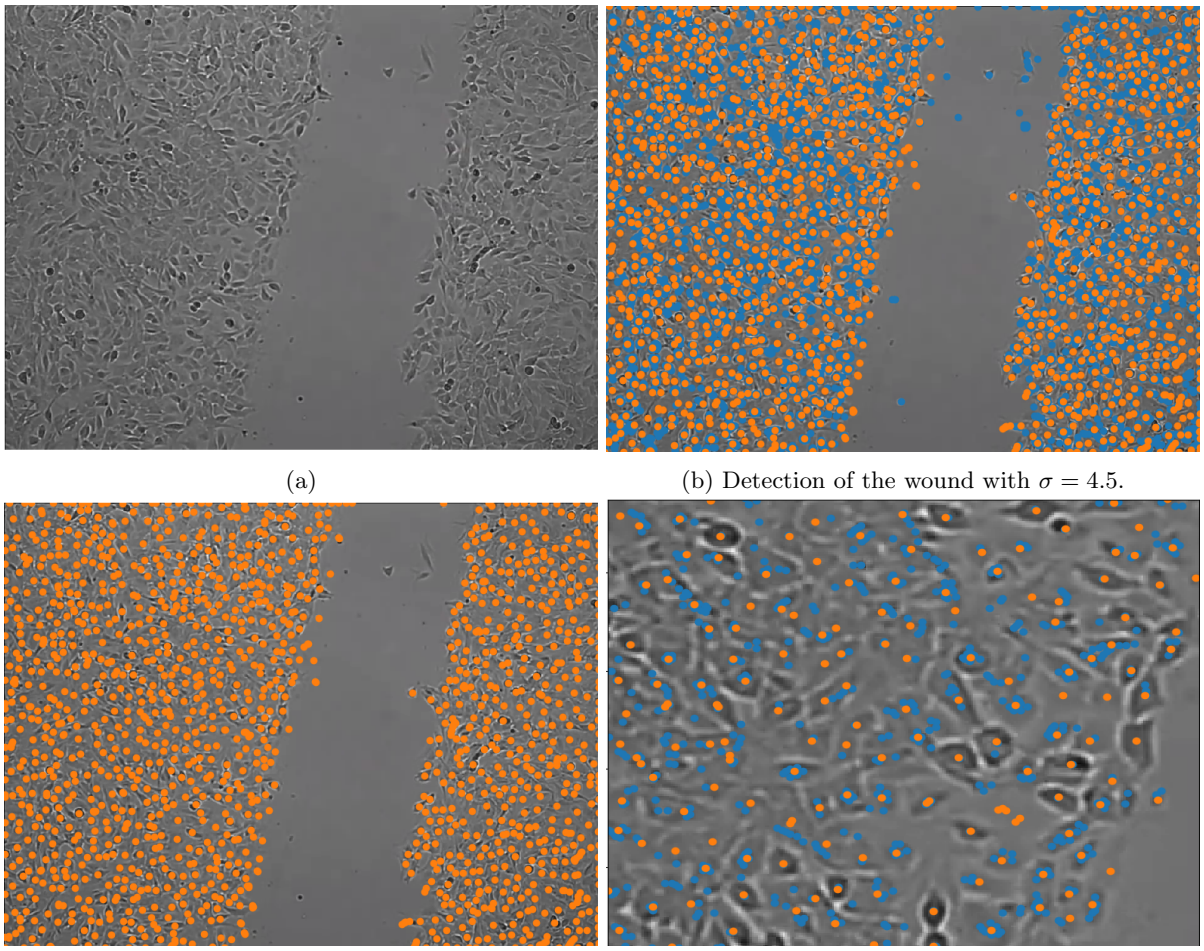
Figure 30: Results from the third video.

Figure 30 are quite interesting. First of all, we see that the method without a Sobel filter has a hard time approximating the area, probably due to poor (pre)processing of the image. Whereas the Sobel plots are more consistent, except for some huge spikes. These spikes come from the fact that the video itself 'blinks' a few times, making the video darker, causing the filter to actually catch the whole frame as the contour, see Figure 30d. But if we would follow the graphs and ignore the spikes, the graphs also look similar to that of Figure 29c and Figure 13d.

[6]https://www.youtube.com/watch?v=c47IWzRo3Ao

6.2 Cell detection

For both videos we will also look at the cell detection, with and without using histogram equalization. Figure 31 shows us the results of cell detection of the second video. Here, the blue dots represent the



(a)

(b) Detection of the wound with $\sigma = 4.5$.

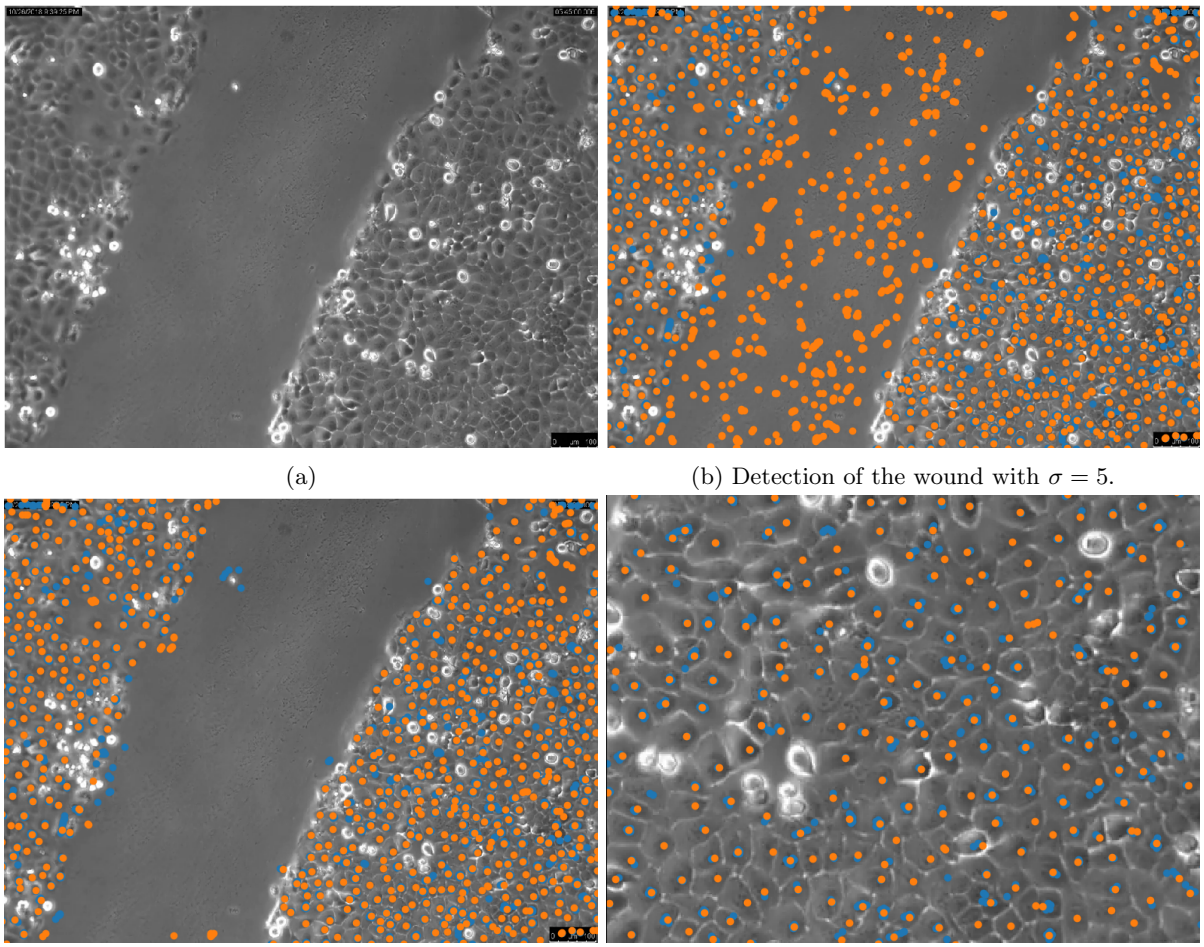
(c) Blue dots removed for clarity.

(d) Zoomed image of the frame.

Figure 31: Results of cell detection from the second video.

method without histogram equalization and orange the opposite. The cells in this video are quite hard to see with our own eyes, since the boundaries are darker. But it should be clear from these figures that there are too many blue dots to be seen (Figure 31b and Figure 31d) compared to the orange ones. For the sake of some clarity, we provided Figure 31c without the blue dots to see how the cell detection worked out with histogram equalization. Similar to previous results of the first video, apart from some cells, most of the cells are detected, and some cells have more than one dot in their interior. The total amount of local maxima here is 1496.

Figure 32 is the result of cell detection in the third video. In Figure 32b, we see that the method with histogram equalization actually detects parts of the wound as well. This is because the wound is not that smooth compared to the previous videos. Instead, there are some places where tiny black dots can be seen, and these dots will be detected. But it is possible to remove the orange dots in the wound, see Figure 32c. Furthermore, in Figure 32d the white cells are not detected, these are the cells which are about to divide. We assumed that the interior of the cells were dark instead of light. So our method was not able to detect them. But again, we were able to catch (almost) all other cells in the frame.



(a)

(b) Detection of the wound with $\sigma = 5$.

(c) Adjusted image.

(d) Zoomed image of the frame.

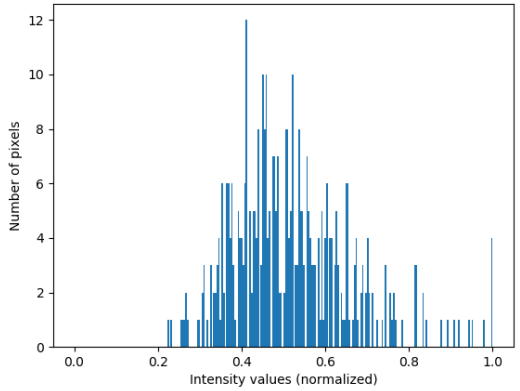
Figure 32: Results of cell detection from the third video.

7 Further research

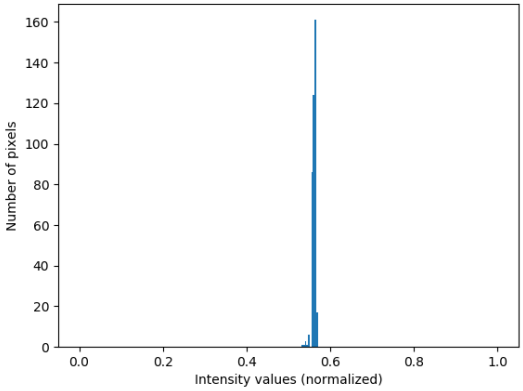
In this last section, we describe two methods where we have been working on it during this project, but they did not work completely or they were not finished yet, probably due to coding errors. Both of these methods have to do with kernel density estimation. The kernel density estimation is a method to smooth the observed data by estimating the probability density function of the given data, for example when one is working with histograms [8]. The first method describes the detection of the boundary and the second method describes a measure to determine the amount of cells in a given image.

7.1 Kernel density estimation for boundary detection

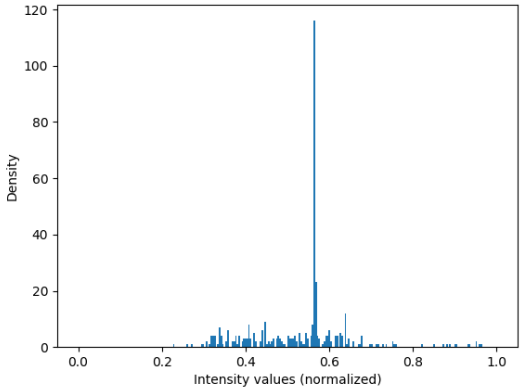
In our attempt to improve the boundary detection of the wound, we have looked at the different histograms of specific regions. In particular, we looked at regions where a group of cells are present, regions of the wound and at the boundary of the wound. Examples of these histograms are shown below, where we normalized the intensity values to [0,1].



(a) Region with cells.



(b) Region of the wound.



(c) Region of the boundary.

Figure 33: Different histograms for different regions of the frame.

Now, the idea of this method is that we want to make three classifications based on the given histograms. These classifications are: 'cells', 'wound' and 'boundary'. So for example, if we create a small window in an image, then based on the histogram of that small window, is it possible to decide whether we are

dealing with 'cells', the 'wound' or the 'boundary'? Eventually, we want to get the pixels which are classified as 'boundary'. The following steps are what we have done to accomplish this method:

- We create a small window in the frame near the boundary and for every pixel in that small window, we look at the histogram within a small area of each pixel.
- Then, we compare the histogram of each pixel with the histogram of the boundary (Figure 33c) by looking at the 'smallest difference' between these histograms.
- Next, the pixel that corresponds to the histogram with the smallest difference will be added to the set of coordinates that corresponds to the boundary of the wound.
- After that, the window will be moved to the coordinates of the pixel which was most recently added, and then repeat this process.

By saying the 'smallest difference', we actually mean the smallest distance between the histograms. This distance can be computed using the ***Euclidean distance*** (also called ***Euclidean metric*** or $\mathbf{L^2}$**-norm**. The Euclidean distance is defined as:

$$\|\mathbf{x}\|_2 = \left(\sum_{k=1}^{n} |x^k|^2\right)^{1/2} = \sqrt{\sum_{k=1}^{n} |x^k|^2}, \quad \mathbf{x} = (x_1, x_2, \ldots, x_n). \tag{7.1}$$

However, to get a better comparison between the histograms, we will look at the corresponding *kernel density estimations* of the histograms. Our histograms in Figure 33 have many intensity values which are not counted at all (discontinuities). As a consequence, calculating the distance between the histograms will have some difficulties.

The definition of the kernel density estimator is as follows:

Definition 7.1. *Let $x_1, x_2, \ldots, x_n$ be the sample of size n from a random variable with an unknown density f. Then the kernel density estimator of f at the point x is defined by*

$$\hat{f}_h = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right), \tag{7.2}$$

where K is the kernel, a non-negative, real-valued integrable function and h is called the bandwidth, a smoothing parameter which is positive-valued. The kernel K_h with subscript h is the scaled kernel given by $K_h = \frac{1}{h}K(\frac{x}{h})$.

The kernel function K satisfies the following conditions:

$$\int_{-\infty}^{\infty} K(u)\, du = 1. \tag{7.3}$$

This condition ensures that the kernel density estimator $\hat{f}_h$ is indeed a probability density function. The second condition is not entirely necessary, but often in practice, the kernel is symmetric about zero. So in this case, the mean of the kernel is also zero:

$$\int_{-\infty}^{\infty} uK(u)\, du = 0. \tag{7.4}$$

There are several functions which satisfy the conditions of a kernel, but we will use the so-called ***Gaussian kernel***, given by:

$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}. \tag{7.5}$$

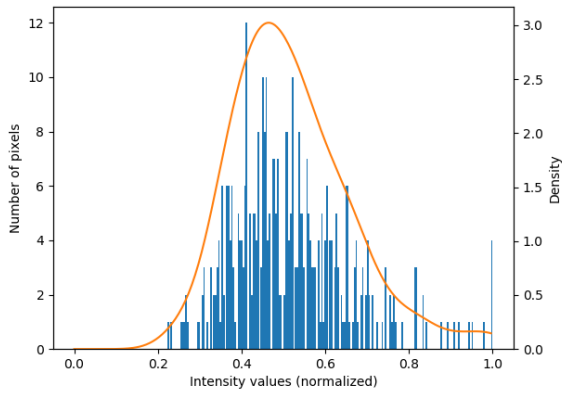
We will use the Gaussian kernel because this kernel is defined for all values of $u \in (-\infty, \infty)$, whereas for many other kernels, they are only defined on $u \in [-1, 1]$ and for all other values of u, $K(u) = 0$. Also,

this kernel is used very often and the Gaussian function is a well-known function in probability theory, so we will stick to this choice too.

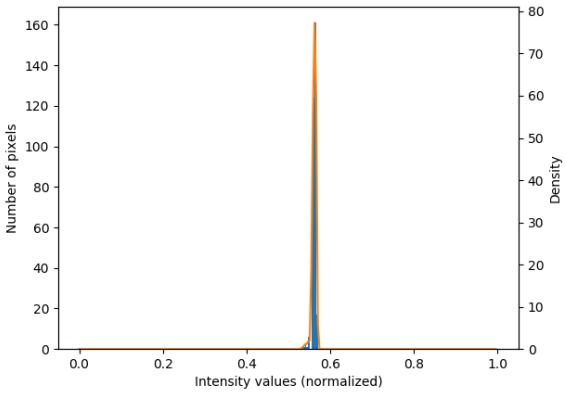
A suitable choice for h needs to be chosen so that the kernel density estimator is a good representation of the real data. In the book written by Silverman [8], a suitable choice for h is given by:

$$h = \left(\frac{4\sigma^5}{3n}\right)^{1/5} \approx 1.06\sigma n^{-1/5}, \tag{7.6}$$

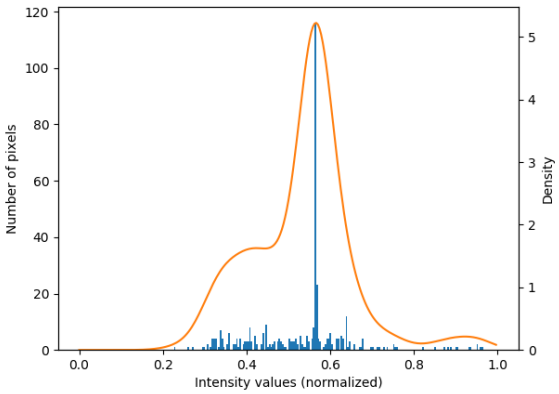
where σ denotes the standard devitation of the data and n the total number of observations. Implementing the kernel density estimator given by (7.2) with the Gaussian kernel (7.5) and with bandwidth h (7.6) will give us the figures in 34. Both the histograms and the kernel density estimation are plotted to clarify



(a) Region with cells.



(b) Region of the wound.



(c) Region of the boundary.

Figure 34: Histograms and their corresponding kernel density estimation.

the shapes of the kernel density estimation. We see that the kernel density estimations fit the data quite well, so that we can work with them instead of the histograms. One should notice though that the density plots and the histograms have different scales on the y-axis. So when the same scales is used for both the histograms and density plots, the density plots will be much smaller than what we see now in Figure 34. Especially Figure 34c looks a bit odd when different scales are used, but if one would use the same scale on the y-axis and zoom in on the plot, then the density plot fits the given data quite well too.

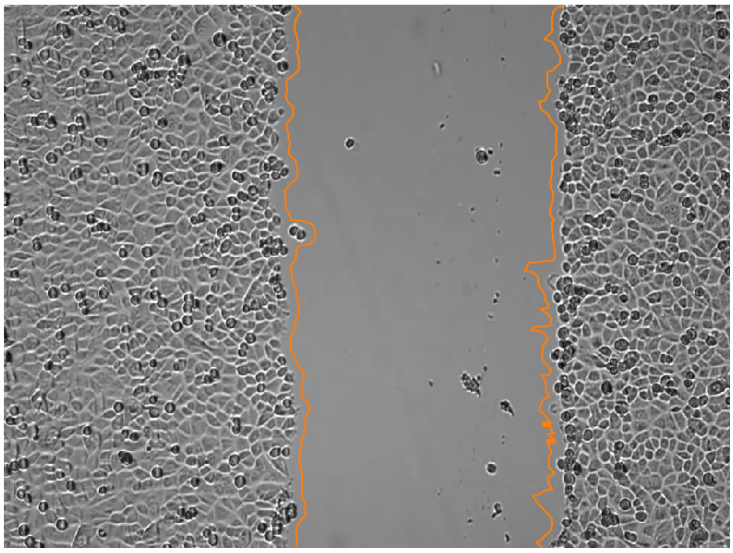


Figure 35: Detection of the wound using KDE.

In Figure 35, we see that the detection of the boundary went quite well. The lines are a bit more inside the wound then on the boundary itself and on the right side, it looks like the detection went a bit rough. However, the overall shape of the wound remains.

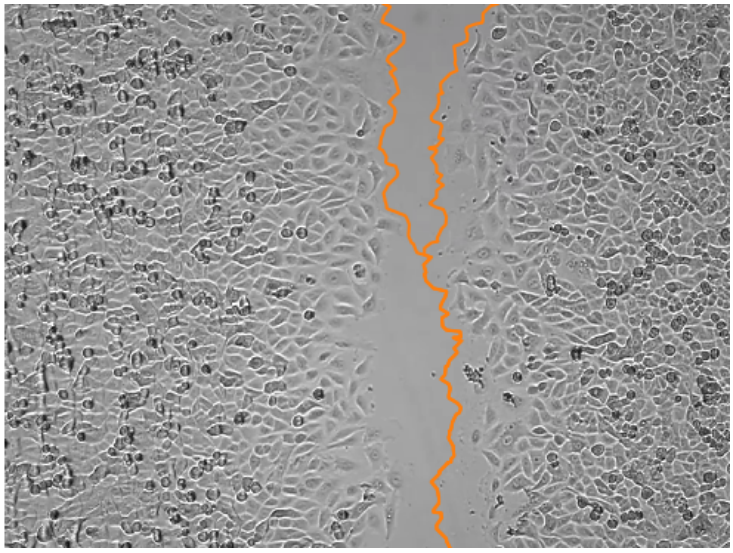


Figure 36: Detection of the wound using KDE on a different frame.

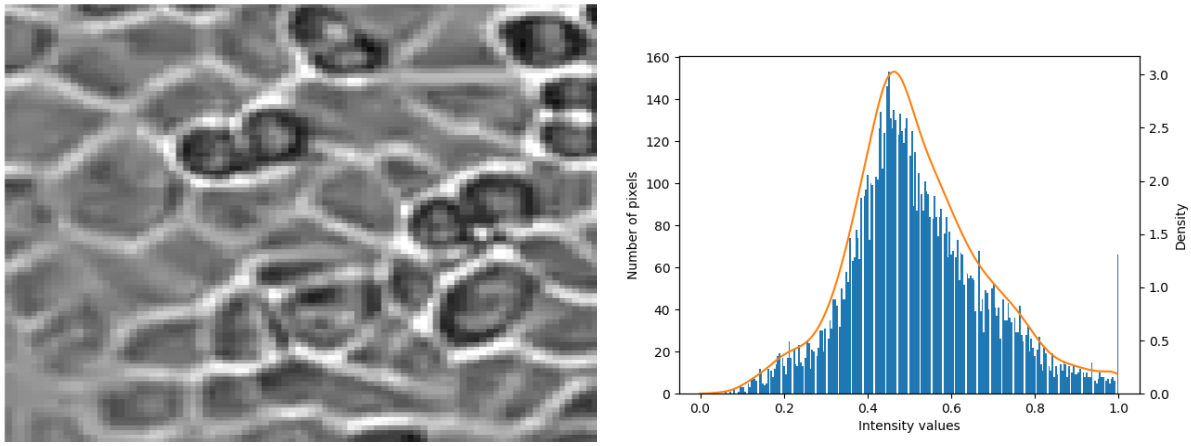
If we look at another frame and apply the same detection, the result is less accurate, see Figure 36. One of the reasons why the detection is not accurate is that we did not update the histogram of the boundary in every frame. Instead, we keep comparing to the histogram given in Figure 34c.

Also, we only compared the histograms of the pixels to the histogram of the boundary. However, it would be better if the pixels are also compared to the histogram of the cells and the wound. In this way, the classification process would be more accurate. In future studies, this can be an alternative way to detect boundaries (of wounds).

7.2 Kernel density estimation for cell detection

In previous sections, we showed how to detect the cells by detecting the local maxima in every frame. But in this section, we will look at a different perspective to measure the amount of cells (or cell density) in an image. This is mostly based on the analysis of the image's histogram and their corresponding kernel density estimation (KDE). We want to be able to say something about the amount of cells (or cell density) purely by looking at the histogram of the image. In this way, we are less dependent on the image that must be analysed.

First, we will show an image that is going to be analysed and the corresponding histogram.



(a) Image.

(b) Histogram and the KDE.

Figure 37: Image to be analysed.

In this image, we counted about 45 cells by hand. Now is it possible to derive a measure for the amount of cells purely from the image's histogram? We see from this image that the boundaries are bright, whereas the interior of the cells are darker. This can be seen from the histogram as well, where most of the pixels are between values 0.4 and 0.6. One way to look at the histogram is to consider the ratio between the pixels that are dark and bright. To define 'dark' and 'bright', we will make use of the kernel density estimator again, given also in Figure 37.

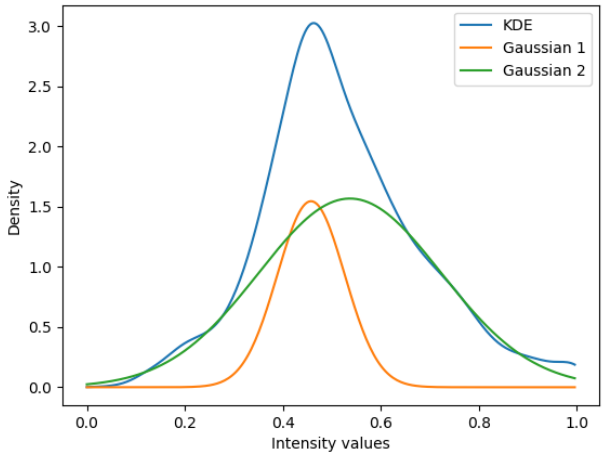


Figure 38: KDE split into two Gaussians.

Because we want to know what the ratio is between the 'dark' and 'bright' pixels, we want to fit the

KDE with two Gaussian curves (the KDE is already a sum of Gaussians), one that represents the 'dark' pixels and the other that represents the 'bright' pixels. We define the Gaussian curves as:

$$f(I) = Ne^{-\alpha(I-I_1)^2} + Me^{-\beta(I-I_2)^2}, \tag{7.7}$$

with the first term representing the 'dark' pixels, and the second term the 'bright' pixels. N, α, I_1, M, β, and I_2 are the parameters to be estimated. Fitting has been done by the function `optimize.curve_fit` from `SciPy`, which uses non-linear least squares to fit the curves with the given parameters, and the results are seen in Figure 38.

The parameters $(N, \alpha, I_1, M, \beta, I_2)$ are given by the (rounded) values (1.55, 106.30, 0.46, 1.57, 14.44, 0.54). Surprisingly, the curve 'Gaussian 2' with (mean) $I_2 = 0.54$ is much wider than the curve 'Gaussian 1'. We actually expected that the curves would be reversed, so that 'Gaussian 1' is much wider than 'Gaussian 2' because there are more dark pixels to be seen (interior of the cells) than bright pixels (the boundaries).

The area under the 'Gaussian 1' curve is approximately 0.27, whereas 'Gaussian 2' has area 0.73 (it should be summed up to 1, because the KDE is a probability density function). So we are actually dealing with some kind of probability. Looking at Figure 38 and their areas, one might say that the probability of a pixel being 'dark' is $p_{dark} = 0.27$, whereas the probability of being 'bright' is then $p_{bright} = 0.73$. But unfortunately, this does not correspond to what we see in the image. We have also tried it for other functions as well, but the results were mixed:

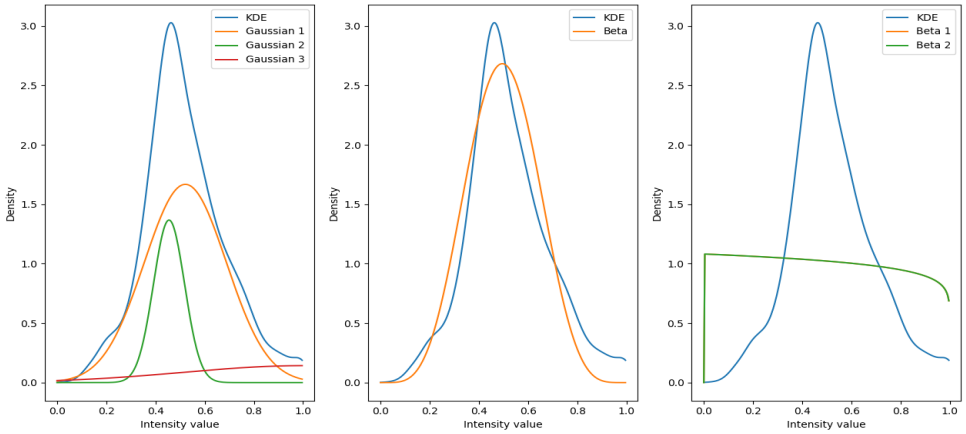


Figure 39: Left: three Gaussian curves plotted. Middle: One Beta distribution. Right: two Beta distributions plotted, but the values of the parameters were assigned the same.

The left figure did not change at all, except for an extra Gaussian curve that is not in the same range as the other Gaussians. The middle and right figures are the Beta distributions, but we were only able to plot a single Beta distribution, whereas plotting the sum of two Beta distributions would give us an interesting curve. Also, using different images give us similar plots such as Figure 38. Perhaps using other distribution functions which are similar to the Gaussian will have a better performance.

If we are able to get the distributions that correspond to the image, we have the probability of being a bright pixel p_{bright} and a dark pixel p_{dark}. Then the area of dark pixels in an image (the area of all the cells) is given by:

$$A_{dark} = p_{dark}A_{image} = (1 - p_{bright})A_{image}. \tag{7.8}$$

Furthermore, if we know the (average) area of a single cell, then the total amount of cells would be:

$$N_{cells} = A_{dark}/A_{cell}. \tag{7.9}$$

Looking back at Figure 2, we know that the large spike in the histogram corresponds to the pixels of the wound. If we can determine a distribution function that corresponds to the spike, we have a probability

that a pixel is in the wound, say p_{wound}. Then in the same way as in equation (7.8), we can determine the area of the wound in pixels:

$$A_{wound} = p_{wound} A_{image}. \tag{7.10}$$

8 Conclusion and discussion

In this thesis, we began by detecting the area of the wound in one of our videos. To detect the wound, we used the open-source package `OpenCV` in Python to draw a contour around the wound. This could only be done in binary images, so frame processing was needed at first to create binary images, such as filtering and eroding the frames. The resulted drawings were quite accurate, and we were able to calculate the corresponding area. But at some video frames, the graph of the area of the wound over time did not correspond entirely to what we saw in the video. Instead of detecting the wound, it detected also some parts of the surrounding tissue as well.

We showed a method to improve the wound detection, by using a different filter called the Sobel filter, which calculates the gradient of the image. The results after the Sobel filter gave us a better representation of what we saw in the video. A drawback of the wound detection is that it can be affected when the wound is almost closed. What happens then is that the wound consists of multiple parts and only the largest part will be detected. A possible solution for this is by extracting the top two or three largest contours that are found in those frames. After that, if we sum up the areas of those contours, then we will have the area of the whole wound. Perhaps if we used a different filter, the results would be even better. There are many filters in the field of image processing which we did not examined, many of them already exist in the Python package `Scikit-image`.

Next, the detection of individual cells has been showed. By blurring the image and localizing the local maxima, we were able to detect almost all of them. But some cells were more than once located, so we would have to much cells detected. Therefore, histogram equalization has been applied on the images to increase the overall contrast of the image before detecting the local maxima. In this way, the detection was more consistent and accurate.

If we could compare our results with other existing software packages that extract the cell density and the area of the wound, that would be interesting to see how well our methods performed. One of the software packages we had in mind was CellProfiler[7]. Unfortunately, we did not have enough time to use this software and compare it with our results.

After that, we applied the continuum model on the video. We derived an approximation of the continuum model by using finite difference. The parameter estimation has been discussed as well to find a suitable parameter to fit the model with the real data. We actually found two suitable parameters D_{LS} and D_{avg}, where the results were similar to the real measurements using these parameters. In the end, a method to use a source term is explained as well. In the same way, it may be possible to determine the optimal parameters for other existing wound-healing models as well.

We have shortly displayed several results of boundary detection and cell detection in two other videos. Most of the results look similar to the results that we obtained at first, but the results were also dependent on the videos themselves.

At last, we described two methods where we have been working on it during this project, but we were not able to finish it or to make it work completely. Both methods are based on the kernel density estimation. The first method was used to detect the boundary of the wound and the second method was used to determine a new measure of the amount of cells in an image.

[7] https://cellprofiler.org/

References

[1] Satoshi Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[2] den Bakker, D. (2018). Analysis of Microscopic Images: A Gradient Vector Flow Based Approach. Retrieved from http://resolver.tudelft.nl/uuid:d34e0179-f911-40f3-94a8-73944cb7e2ff

[3] Adam J.A. (1999). *A simplified model of wound healing (with particular reference to the critical size defect).* Math Comput Model 30(5–6):23–32

[4] *Histogram Equalization.* Retrieved from https://www.math.uci.edu/icamp/courses/math77c/demos/hist_eq.pdf

[5] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*, Third Edition, page 91-93, 2008.

[6] Vermolen, F.J. and Pölönen, I. (2019). *Uncertainty quantification on a spatial Markov-Chain model for the progression of skin cancer.*

[7] Snowden, J.M. (1983). *Wound Closure: An Analysis of the Relative Contributions of Contraction and Epithelialization.*(pp.453-463).

[8] Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis.*(pp.44-45) Chapman and Hall, London.

[9] Sherratt, Jonathan A., and J. D. Murray. "Models of epidermal wound healing." *Proceedings of the Royal Society of London B: Biological Sciences* 241.1300 (1990): 29-36.

A Python Code

In this section, the most important codes are found. Throughout this project, we worked with Python 3.7.3. Many open-source libraries such as `OpenCV`, `SciPy` and `Numpy` were used.

A.1 Boundary detection

```
import cv2
import scipy.ndimage as nd
import numpy as np
import matplotlib.pyplot as plt
import scipy
from scipy.spatial import distance as dist
from skimage import util

cap = cv2.VideoCapture('video_file')

frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

## ========================== Boundary wound detection ======================== ##
def boundary_sobelx(im): #Boundary detection using Sx filter
    gray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray,(5,5),3)
    sx = nd.sobel(blurred,axis = 0, mode = 'constant')
    invert_x = util.invert(sx)
    dif_x = invert_x - sx
    ret3, newthresh  = cv2.threshold(dif_x,254,255,cv2.THRESH_BINARY)
    dilate = cv2.dilate(newthresh,None,iterations = 2)
    contours, hierarchy = cv2.findContours(dilate.copy(), cv2.RETR_EXTERNAL,
    ↪  cv2.CHAIN_APPROX_SIMPLE)
    B_array = max(contours,key = len)                    #finds the largest contour
    img = cv2.drawContours(im,B_array,-1, (0,0,255),5)   #draws contour on frame
    return B_array

def boundary_sobely(im): #Boundary detection using Sy filter
    gray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray,(5,5),3)
    sy = nd.sobel(blurred,axis = 1, mode = 'constant')
    invert_y = util.invert(sy)
    dif_y = invert_y - sy
    ret0, newthresh  = cv2.threshold(dif_y,254,255,cv2.THRESH_BINARY)
    dilate = cv2.dilate(newthresh,None,iterations = 2)
    contours, hierarchy = cv2.findContours(dilate.copy(), cv2.RETR_EXTERNAL,
    ↪  cv2.CHAIN_APPROX_SIMPLE)
    B_array = max(contours,key = len)                    #finds the largest contour
    img = cv2.drawContours(im,B_array,-1, (0,0,255),5)   #draws contour on frame
    return B_array

def boundary_sobel(im): #Boundary detection using combined filter
    gray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray,(5,5),3)
    sx = nd.sobel(blurred,axis = 0, mode = 'constant')
```

```
44          sy = nd.sobel(blurred,axis = 1, mode = 'constant')
45          sobel = abs(sx) + abs(sy)
46          invert_s = util.invert(sobel)
47          dif_s = invert_s - sobel
48          ret4, newthresh  = cv2.threshold(dif_s,254,255,cv2.THRESH_BINARY)
49          dilate = cv2.dilate(newthresh,None,iterations = 2)
50          contours, hierarchy = cv2.findContours(dilate.copy(), cv2.RETR_EXTERNAL,
            ↪  cv2.CHAIN_APPROX_SIMPLE)
51          B_array = max(contours,key = len)                    #finds the largest contour
52          img = cv2.drawContours(im,B_array,-1, (0,0,255),5)   #draws contour on frame
53          return B_array
54
55      def boundary(im): #no sobel filter
56          gray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
57          blurred = cv2.GaussianBlur(gray,(5,5),3)
58          ret1, threshblur1 = cv2.threshold(blurred,130,255,cv2.THRESH_BINARY)
59          ret2, threshblur2 = cv2.threshold(blurred,92,255,cv2.THRESH_BINARY)
60          newblur = threshblur2 - threshblur1
61          erode = cv2.erode(newblur,None,iterations = 2)
62          contours, hierarchy = cv2.findContours(erode.copy(), cv2.RETR_EXTERNAL,
            ↪  cv2.CHAIN_APPROX_SIMPLE)
63          B_array = max(contours, key = len)                       #finds the largest
            ↪  contour
64          img = cv2.drawContours(im,B_array,-1, (0,0,255),5)  #draws contour on frame
65          return B_array
66
67      def area(curve): #Green's Theorem on calculating area
68      n=0
69      res = 0
70      for n in range(len(curve)-1):
71          res = res + (1/2)*(curve[n+1][0]+curve[n][0])*(curve[n+1][1] - curve[n][1])
72          n = n+1
73      return abs(res)
74
75      ##============================== Area calculation ==============================##
76
77      def calc_area(k): #calculating area contour until frame number k
78          B = {}
79          B_x = {}
80          B_y = {}
81          B_s = {}
82          Area_x = []
83          Area_y = []
84          Area = []
85          Area_s = []
86          for i in np.arange(k):
87              ret,frame = cap.read()
88              if ret:
89                      aa = boundary(frame)
90                      aa = np.squeeze(np.array(aa))
91                      B[str(i)] = aa
92                      aaclose = np.insert(aa,0,aa[len(aa)-1],0) #make contour closed
93                      Area = Area +[area(aaclose)]
94
95                      #sobel x
```

```
                bb = boundary_sobelx(frame)
                bb = np.squeeze(np.array(bb))
                B_x[str(i)] = bb
                bbclose = np.insert(bb,0,bb[len(bb)-1],0) #make contour closed
                Area_x = Area_x +[area(bbclose)]

                #sobel y
                cc = boundary_sobely(frame)
                cc = np.squeeze(np.array(cc))
                B_y[str(i)] = cc
                ccclose = np.insert(cc,0,cc[len(cc)-1],0) #make contour closed
                Area_y = Area_y +[area(ccclose)]

                #sobel
                dd = boundary_sobel(frame)
                dd = np.squeeze(np.array(dd))
                B_s[str(i)] = dd
                ddclose = np.insert(dd,0,dd[len(dd)-1],0) #make contour closed
                Area_s = Area_s +[area(ddclose)]

        i = i+1
    cap.release()
    cv2.destroyAllWindows()
    return Area, Area_x, Area_y, Area_s
```

A.2 Cell detection

```
import cv2
import scipy.ndimage as nd
import numpy as np
import matplotlib.pyplot as plt
import scipy
import scipy.ndimage.filters as filters

cap = cv2.VideoCapture("video_file")
ret, frame = cap.read()

def findMaxima(I,sigma): #cell detection without histogram equalization
    invert = np.invert(I)
    size = 5
    threshold = 2
    im_gauss = nd.gaussian_filter(invert,sigma)
    im_gauss_min = filters.minimum_filter(im_gauss, size)
    im_gauss_max = filters.maximum_filter(im_gauss, size)
    maxima = (im_gauss == im_gauss_max)
    diff = ((im_gauss_max - im_gauss_min)>threshold)
    maxima[diff == 0] = 0

    labeled, num_objects = nd.label(maxima)
    slices = nd.find_objects(labeled)
    x, y = [],[]
    for dy, dx, dz in slices:
        x_center = (dx.start + dx.stop - 1)/2
        x.append(x_center)
        y_center = (dy.start + dy.stop - 1)/2
        y.append(y_center)
    return x,y

def findMaxima_eq(I,sigma): #cell detection with histogram equalization
    gray_frame = cv2.cvtColor(I,cv2.COLOR_BGR2GRAY)
    equ = cv2.equalizeHist(gray_frame) #Histogram equalization, higher contrast
    invert_equ = np.invert(equ)
    size = 5
    threshold = 2
    im_gauss = nd.gaussian_filter(invert_equ,sigma)
    im_gauss_min = filters.minimum_filter(im_gauss, size)
    im_gauss_max = filters.maximum_filter(im_gauss, size)
    maxima = (im_gauss == im_gauss_max)
    diff = ((im_gauss_max - im_gauss_min)>threshold)
    maxima[diff == 0] = 0

    labeled, num_objects = nd.label(maxima)
    slices = nd.find_objects(labeled)
    x, y = [],[]
    for dy, dx in slices:
        x_center = (dx.start + dx.stop - 1)/2
        x.append(x_center)
        y_center = (dy.start + dy.stop - 1)/2
        y.append(y_center)
```

```
53          return x,y
54
55      def count_cells(I,sigma):
56          imshape_x, imshape_y = I.shape[:2]
57          num_lines =  round(imshape_y/10) # number of rows and columns used to compute
            ↪  S(I)
58          im_gauss = nd.gaussian_filter( I, sigma)
59          avg_x_cells , avg_y_cells = 0, 0
60          for c in range ( num_lines ):
61              n = c* imshape_x / num_lines
62              m = c* imshape_y / num_lines
63              line_x = im_gauss[int(n),:,0]
64              line_y =im_gauss[:,int(m),0]
65              line_x_mean = line_x.mean ()
66              line_y_mean = line_y.mean ()
67              boundary_x = ( line_x < line_x_mean )
68              boundary_y = ( line_y < line_y_mean )
69              switch_x = abs(np.diff( boundary_x ))
70              switch_y = abs(np.diff( boundary_y ))
71              count_x = switch_x.sum() *0.5
72              count_y = switch_y.sum() *0.5
73              avg_x_cells += count_x / num_lines
74              avg_y_cells += count_y / num_lines
75          num_cells_scan = avg_x_cells * avg_y_cells
76          return num_cells_scan
```

A.3 Continuum model

```
import cv2
import scipy.ndimage as nd
import numpy as np
import matplotlib.pyplot as plt
import scipy
import scipy.sparse as sp
import scipy.sparse.linalg as spla

#setting up grid
h = 10
a = 80
M = 88
N = 64
## ====================== Real measurement ====================##
def measured_data(frame_number): #measured data at frame number
    u2d = []
    cap.set(cv2.CAP_PROP_POS_FRAMES,frame_number)
    _,frame = cap.read()
    for j in range(0,N+1):
        u = []
        for i in range(0,M+1):
            window = frame[(h*j): a +(h*j),(h*i):a +(h*i)]
            u += [count_cells(window,0.7)]
        u2d += [u]
    u2d = np.array(u2d)
    img = u2d.reshape((j+1,i+1))[1:j,1:i]
    u_vec = img.ravel() #vector of measured data lexographically
    return u_vec, img

## ================== Laplacian matrix ==================== ##
Q = sp.diags([1,-4,1],[-1,0,1], shape = (M-1,M-1))
block = np.kron(np.eye(N-1),Q.toarray())
l = sp.diags([1,1],[-(M-1),(M-1)],shape=((M-1)*(N-1),(M-1)*(N-1))).toarray()
A = l + block

## ================== Boundary values ====================##
def boundary_west_east(frame_number): #boundary value at frame number
    bdy_west_east = []
    cap.set(cv2.CAP_PROP_POS_FRAMES,frame_number)
    _,frame = cap.read()
    bdy = []
    for j in range(1,N):
        u2 = []
        for i in range(0,M+1,M):
            window = frame[(h*j):a +(h*j),(h*i):a+(h*i)]
            u2 += [count_cells(window,0.7)]
        bdy += [u2]
    nul = np.zeros((i-1)*(j))
    for k in range(0,j):
        nul[(i-1)*k] = bdy[k][0]
        nul[(i-1)*k + (i-2)] = bdy[k][1]
    bdy_west_east = nul
```

```
53          return bdy_west_east
54
55      def boundary_north_south(frame_number): #boundary value at frame number
56          bounds = []
57          bdy_north = []
58          bdy_south = []
59          bdy_vec = []
60          cap.set(cv2.CAP_PROP_POS_FRAMES,frame_number)
61          ret,frame = cap.read()
62          for j in np.arange(0,N+1,N):
63              u = []
64              for i in range(0,M+1):
65                  window = frame[(h*j):a+(h*j),(h*i):a+(h*i)]
66                  u += [count_cells(window,0.7)]
67              bounds += [u]
68          bdy_north = bounds[-2][1:i]
69          bdy_south = bounds[-1][1:i]
70          bdy_vec = np.concatenate([bdy_north,np.zeros((i-1)*(j-3)),bdy_south])
71          return bdy_vec
72
73      ##================ Finite Difference model ==================##
74      def model_data(k,D):
75          u_model,_ = measured_data(0) #begin data
76          it = 0
77          while it < k:
78              bwe = boundary_west_east(it)
79              bns = boundary_north_south(it)
80              u_model = u_model + ((dt*D)/((h)**2))*(A.dot(u_model))
                ↪  +((dt*D)/(h**2))*(np.array(bns) +np.array(bwe))
81              it += 1
82          img_model = u_model.reshape((63,87))
83          return u_model,img_model #solution vector and img
84
85      ## ===================== D_k calculation ======================##
86      def Dif(k):
87          u_k_1,_ = measured_data(k + 1)
88          u_k,_ = measured_data(k)
89          bwe = boundary_west_east(k)
90          bns = boundary_north_south(k)
91          v_k = (dt/(h**2))*(bns+bwe)
92          B = (dt/(h**2))*A
93          b_k = - u_k_1.dot(B.dot(u_k)) - v_k.dot(u_k_1) + u_k.dot(B.dot(u_k)) +
            ↪  u_k.dot(v_k)
94          a_k = v_k.dot(v_k) + + v_k.dot(u_k) + (B.dot(u_k)).dot(B.dot(u_k)) +
            ↪  (B.dot(u_k)).dot(v_k)
95          D_k = - (b_k/(a_k))
96          return D_k, b_k,a_k
97
98      ## ================ Function used to count cells =============##
99      def count_cells(I,sigma): #sigma = 0.7
100         imshape_x, imshape_y = I.shape[:2]
101         num_lines =  round(imshape_y/10) # number of rows and columns used to compute
            ↪  S(I)
102         im_gauss = nd.gaussian_filter( I, sigma)
103         avg_x_cells , avg_y_cells = 0, 0
```

```
104         for c in range ( num_lines ):
105             n = c* imshape_x / num_lines
106             m = c* imshape_y / num_lines
107             line_x = im_gauss[int(n),:,0]
108             line_y =im_gauss[:,int(m),0]
109             line_x_mean = line_x.mean ()
110             line_y_mean = line_y.mean ()
111             boundary_x = ( line_x < line_x_mean )
112             boundary_y = ( line_y < line_y_mean )
113             switch_x = abs(np.diff( boundary_x ))
114             switch_y = abs(np.diff( boundary_y ))
115             count_x = switch_x.sum() *0.5
116             count_y = switch_y.sum() *0.5
117             avg_x_cells += count_x / num_lines
118             avg_y_cells += count_y / num_lines
119         num_cells_scan = avg_x_cells * avg_y_cells
120         return num_cells_scan
```

A.4 Kernel density estimation

```
import cv2
import scipy.ndimage as nd
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage as nd
import scipy
from scipy.optimize import curve_fit
from scipy.integrate import quad
from scipy.special import gamma

cap = cv2.VideoCapture("video_file")
ret, frame = cap.read()



##==================== Kernel Density Estimation ==============##

def K(u): #Gaussian kernel
 return (1/np.sqrt(2*np.pi))*np.exp(-0.5*(u**2))

def kde(x,norm_hist): #KDE
    som = 0
    h = 1.06*norm_hist.ravel().std()*((len(norm_hist.ravel())))**(-1/5)
    if h > 0:
        for i in range(len(norm_hist.ravel())):
            som = som + K((x - norm_hist.ravel()[i])/h)
    res = som/(len(norm_hist.ravel()) * h)
    return res

##======================== Histograms ========================##

## Histogram wound
crop = frame[500:520,500:520]
wound_gray = cv2.cvtColor(crop,cv2.COLOR_BGR2GRAY)
f_wound = kde(t, (wound_gray)/255)

## Histogram cells
crop2 = frame[200:220,200:220]
cells_gray = cv2.cvtColor(crop2,cv2.COLOR_BGR2GRAY)
f_cells = kde(t, (cells_gray)/255)

## Histogram boundary
crop3 = frame[545:565, 375:395]
bound_gray = cv2.cvtColor(crop3,cv2.COLOR_BGR2GRAY)
f_bound = kde(t,(bound_gray)/255)

def kde_boundary(I,l,r):
    c = boundary_sobely(I)
    c = np.squeeze(np.array(c))
    coord_left = np.array([[l,r]]) #begin
    while coord_left[-1][0] < I.shape[0]:
        dic = {}
```

```
            for i in range(1,11):
                for j in range(0,21):
                    if coord_left[-1][1]+j> I.shape[1] or coord_left[-1][0]+i+10 >
                    ↪  I.shape[0]:
                        break
                    else:
                        zoom = I[int(coord_left[-1][0]+i):\
                                 int(coord_left[-1][0]+i+5),int(coord_left[-1][1]+j-20)\
                                 :int(coord_left[-1][1]+j)]
                        zoom_gray = cv2.cvtColor(zoom,cv2.COLOR_BGR2GRAY)
                        f_zoom = kde(t, (zoom_gray)/255)
                        dic[i,j]= [np.linalg.norm(f_grens - f_zoom)] #L2-norm
            for key, val in dic.items():
                if val == max(dic.values()):
                    coord = key
            new_coord_left = np.array((coord_left[-1][0] +
            ↪  coord[0],(((coord_left[-1][1]+coord[1])-20) +
            ↪  coord_left[-1][1]+coord[1])/2))
            coord_left = np.vstack((coord_left,new_coord_left))
        return coord_left

    ## ================== Cell measurement ================== ##

    t = np.arange(0,1,1/256)
    zoom = frame[250:350,220:310]
    zoom_gray = cv2.cvtColor(zoom,cv2.COLOR_BGR2GRAY)
    ydata = kde(t,(zoom_gray/255))

    #define function with two Gaussians
    def func(I,N,a,I_1,M,b,I_2):
        return N*np.exp(-a*(I - I_1)**2) + M*np.exp(-b*(I - I_2)**2)

    def func2(I,N,a,I_1):
        return N*np.exp(-a*(I - I_1)**2)

    def Beta(a,b):
        return (gamma(a)*gamma(b))/gamma(a+b)

    def Beta_pdf2(I,a,b,c,d): #Beta distribution
        return (1/Beta(a,b))* (I**(a-1))*((1-I)**(b-1)) +\
               (1/Beta(c,d))* (I**(c-1))*((1-I)**(d-1))

    def Beta_pdf(I,a,b):
        return (1/Beta(a,b))* (I**(a-1))*((1-I)**(b-1))

    popt,pcov = curve_fit(func, t,ydata, bounds =
    ↪  (0,[np.inf,np.inf,1,np.inf,np.inf,1,np.inf,np.inf,1]))

    curve1 = func2(t,popt[0],popt[1],popt[2])
    curve2 = func2(t,popt[3],popt[4],popt[5])

    ##integration of function
    ans, err = quad(func2,0,1,args=(popt[0],popt[1],popt[2]))
```