Faculteit Bètawetenschappen

Structure based calculation of anisotropy in homo-FRET

Bachelor thesis

Author:
Sander Gerringa

Study: Physics & Astronomy

Supervisor:
Gerhard A. Blab
Utrecht University
Molecular Biophysics group,
Debye institute for nanomaterials

June 2016

Abstract

In this thesis the influence of a structure based method on anisotropy calculation of a biological membrane will be discussed. A simulation of a homo-FRET microscopy experiment was made in which a simple biological cell is approximated. This was used to compare the effects of different gridsizes of a structure used to analyse the cell, as well as different intervals of the individual grids and the influence of the inner cell. This was done on misaligned images to see if this method could omit the need for perfect image registration in the experiment. While the method does seem promising, it does invoke a small deviation with respect to the pixel by pixel calculation. This might be solved by a better treatment of the G-factor of the setup in the simulation.

Contents

1 Introduction

In microscopy of biological samples it is a common trick to attach a fluorophore to said sample, making it easy to image through fluorescence microscopy. Due to light having a limited resolution limit a number of techniques to overcome this have been found, one of them being Förster resonance energy transfer (FRET). This technique uses the fact that fluorphores in close proximity to one another are capable of energy transfer to one another instead of emitting it directly as light. If the fluorophores would be clustered together multiple instances of this process can take place, thus it is possible to use homo-FRET measurements as an indication of cluster sizes.

Since homo-FRET can only be measured through a (de)polarisation of emitted light it turns out to be vitally important to have correct image registration of two images of different polarisation if we use a pixel by pixel calculation.It is believed that by using a structure based method as opposed to the pixel by pixel calculation the need for perfect image registration can be reduced. The aim of this research is to characterize a structure based method for the calculation of the anisotropy of the polarisation and compare it to the pixel based method.

2 Theory

2.1 Fluorescence microscopy

A flurophore is a chemical that can be excited and after excitation re-emit the excitation energy as light. These chemicals exist in many varieties most of them emitting at a different wavelength. In fluorescence microscopy these chemicals are attached to a sample of interest to be able to image them.

2.2 FRET

Förster resonance energy transfer (FRET) is a process of non radiative energy transfer between two light sensitive molecules through an electrostatic dipole-dipole interaction[1]. An excited molecule can transfer its excitation energy to another molecule, thus losing its energy in a non radiative process and exciting the other molecule ,see figure 1. The efficiency of this process is strongly dependant on the distance between the two molecules ($\propto 1/R^6$)[1]. This allows FRET to be used as a distance measurement. These distances are very small (approximately 1-10 nm) allowing to image way beyond the resolution limit of the light from the fluorophores.

Homo-FRET is FRET between two molecules of the same type and can only be measured by the (de)polarisation of the emitted light[2]. The dipoles in molecules are randomly oriented, when excited with polarised light only the molecules with a dipole oriented in the same direction as the polarisation will be excited. For FRET the orientation of the dipoles does not matter, so if energy is transferred it could transfer to a differently oriented fluorophor. If sufficiently slow rotating fluorophors are used in comparison to the fluorescence lifetime then if no FRET occurs polarisation of the emitted light should be conserved, while if FRET occurs the polarisation of the emitted light is changed (since the molecules are randomly oriented).

2.3 Anisotropy

To quantify the change in polarisation of the emitted light it is common to use the anisotropy (r). r is given by equation 1 [3].

$$r = \frac{I_{\|} - GI_{\perp}}{I_{\|} + 2GI_{\perp}} \tag{1}$$

Here $I_{\|}$ stands for the parallel intensity and $I_{\perp}$ stands for the perpendicular intensity, both with respect to the polarisation of the excitation beam. G stands for the G-factor, which is a correction factor to account for the fact that in an experimental setup, light of different polarisation is treated differently by the same optical elements. Thus one polarisation could be blocked slightly by a lens while the other does not. The 2 in the denominator is a normalisation factor to account for the fact that there are two directions perpendicular to the excitation beam. This makes the denominator the total intensity, while the numerator is the difference in intensity of the different polarisations.

An increase in depolarisation means an increase in $I_{\perp}$, which means a drop in r. Thus a drop in anisotropy is correlated to an increase in depolarisation.

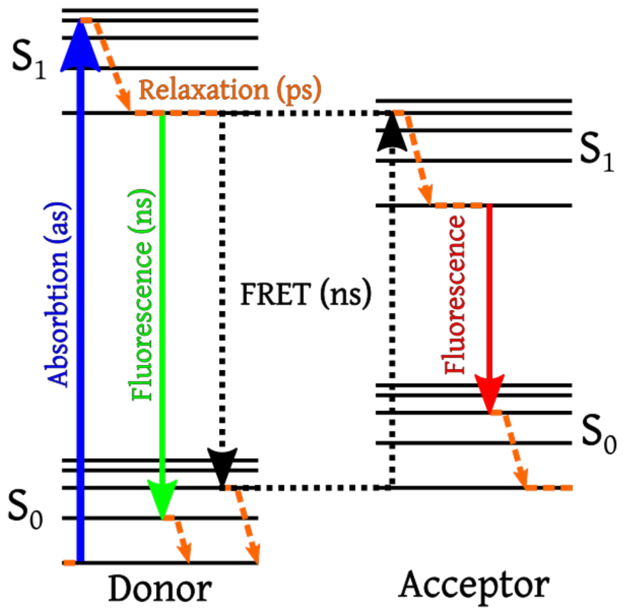


Figure 1: Jablonski diagram showing the concept of FRET, absorbtion occurs in the donor molecule (blue), then either the acceptor molecule is not close enough for FRET to occur and the energy is emitted by regular fluorescence (green) or the acceptor is close enough and FRET occurs and the acceptor molecule emits the energy through fluorescence (red).Figure by Alex M. Mooney,licensed under CC BY-SA 3.0 via Wikimedia Commons.

2.4 Typical experimental setup

In a typical homo-FRET experiment a setup as seen schematically in figure 2 could be used. The excitation source is a mercury lamp, of which the light is being polarised and passed through an excitation filter and a dichroic mirror to the sample. The emitted light from the sample passes through the dichroic mirror and an emission filter before passing into a polarizing beam splitter. Then both polarisations of light are being detected with a camera. The beam splitter is much more efficient in the parallel direction, so part of the parallel beam does leak through into the perpendicular beam. To correct this behaviour another polarizer is placed between the beam splitter and the camera detecting the perpendicular beam. The two cameras are connected with a trigger so they record an image at the same time. The cameras are capable of up to 2560x2560 images, but since biological cells usually are not that big and to save time and memory, usually a smaller image of 512x512 pixels is recorded. It has to be noted that the G-factor is in general not a single number for the entire setup but rather a different number for each pixel of the camera.

2.5 Modeling the setup

An actual experiment would involve imaging a biological cell with a membrane in which a high concentration of fluorophores would be clustered and an inside with a lower concentration of clustered fluorophores see for example figure 3. The cell is modeled by a polygon, and its membrane by the sides of the polygon. To accurately determine which pixels belong to the cell in which amount an algorithm to create three masks (inside,outside and membrane) was made. This required a theoretical treatment of the ways a line (the membrane) can intersect and divide a pixel.

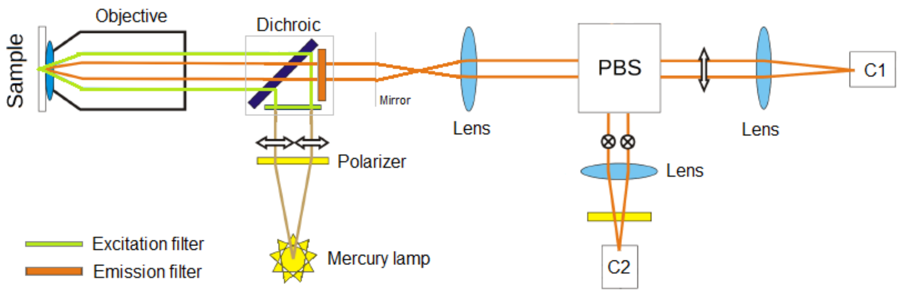


Figure 2: A schematic representation of a typical setup. Excitation light is drawn in green, emission light is drawn in orange. PBS is the Polarising beam splitter, C1 and C2 are the cameras. Figure adapted from [6]

2.5.1 Lines through pixels[1]

First we define a line using $L = \{(x, y)|ax + by + c = 0\}$. This leads to a definition for a (normal) distance from the line d

$$d(ax + by + c = 0, (x_0, y_0)) = \frac{ax_0 + by_0 + c}{\sqrt{a^2 + b^2}} \tag{2}$$

This definition generates both negative and positive distances, depending on whether the point of interest is to the left or the right relative to the direction of the line. The coëfficients of a line passing through two points P_l and P_m are found through

$$\begin{aligned} a_{lm} &= \frac{y_l - y_m}{S} \\ b_{lm} &= -\frac{x_l - x_m}{S} \\ c_{lm} &= -a_{lm}x_l - b_{lm}y_l \\ S &= \sqrt{(y_l - y_m)^2 + (x_l - x_m)^2} \end{aligned} \tag{3}$$

From these coëfficients the angle of the line can simply be calculated from $\tan\alpha = \frac{a_{lm}}{b_{lm}}$.

Now that we have the definition of a line we can consider the ways a line can intersect and divide a pixel and find the corresponding length L and the ratio of how much of the pixel is inside and outside the polygon modeling the cel. A pixel is hereby considered to be a square of size 1x1 surrounding a position (i, j). So $x \in [i - 0.5, i + 0.5[$ is contained within the pixel with first index i. The simplest way to handle this problem is by considering the symmetry of the problem. We start by only considering distances which are positive ($d > 0$). We can do this since $d < 0$ does not change any of the values, just our definition of what we would call inside and outside of the

[1]The theoretical work on this was done with help from G.A. Blab and can be found in full in [4]

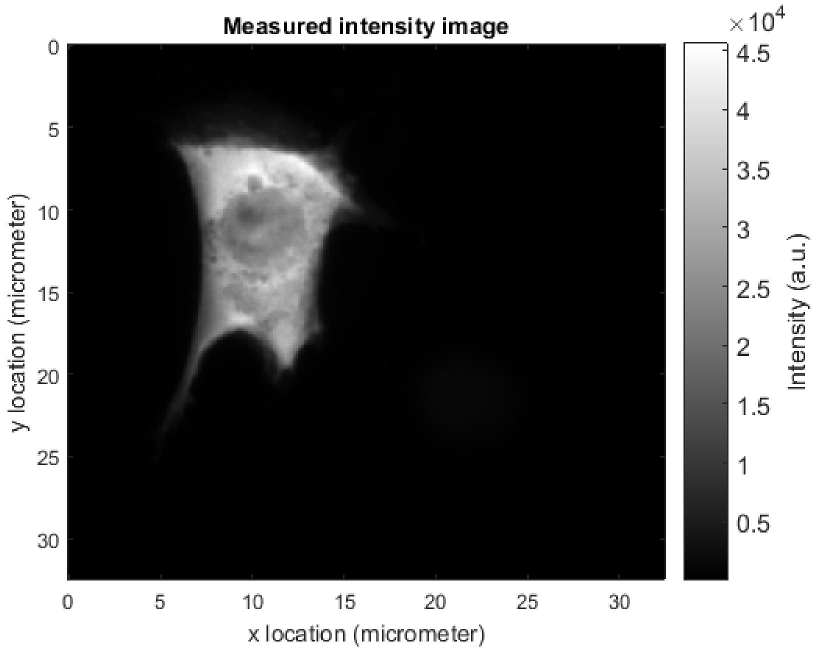


Figure 3: Example of an image of a cell recorded with an actual setup. Areas with higher intensity are found towards the edge of the cell, while the inside has a lower intensity. Image recorded by Kyo Beyeler.

polygon. If we then also limit the angle of the line mod($\frac{\pi}{2}$) we find five possible configurations of the line intersecting a pixel, see figure 4. A further reduction is obtained when another symmetry around $\alpha = \frac{\pi}{4}$ is considered. This leaves only 2 cases, A and C. In case A the area above the line is given by: $A_{above} = 0.5 + \frac{d}{\cos\alpha}$, the area below the line is given by: $A_{below} = 0.5 - \frac{d}{\cos\alpha}$ and the length of the line through the pixel is given by: $L = \frac{1}{\cos\alpha}$, see figure 5. For case C the area below the line is given by: $A_{below} = \frac{hL}{2} = \frac{h^2}{\sin(2\alpha)}$, the area above is then given by:$A_{above} = 1 - A_{above}$ while the length of line through the pixel is given by $L = \frac{2h}{\sin(2\alpha)}$, see figure 6.

This leaves one special case to be considered, the case of a vertex of the polygon being located inside a pixel. In this case we have an inter pixel polygon Q ,see figure 7. If we know all the M vertex points of this inter pixel polygon it's area can be found through[5]:

$$A = \frac{1}{2}\left|\sum_{i}^{M}(x_i y_{i+1} - x_{i+1} y_i)\right|$$

The vertex points situated on the corners of the pixel are easily found, as well as the internal point, since it is a vertex point of the polygon used to model our cell. The first and last point after the internal point can be found by using a parametric definition of a line $P = P_0 + \lambda\Delta p$ with P_0 being the internal point, $\Delta p = [a_{lm}, b_{lm}]$. Now λ can be found through solving:

$$\lambda = -\frac{ap_{o,x} + bp_{o,y} + c}{a\Delta_{P,x} + b\Delta_{P,y}}$$

Now the smallest non-negative λ gives the desired vertex point of the internal polygon. Note that for the last vertex point (Q_5) the direction (Δp) has to be reversed. There are still two special cases to be considered in this, as a λ of 0 implies that the internal point is located on the border of a pixel, the denominator may be zero if two parallel lines are located inside the pixel.

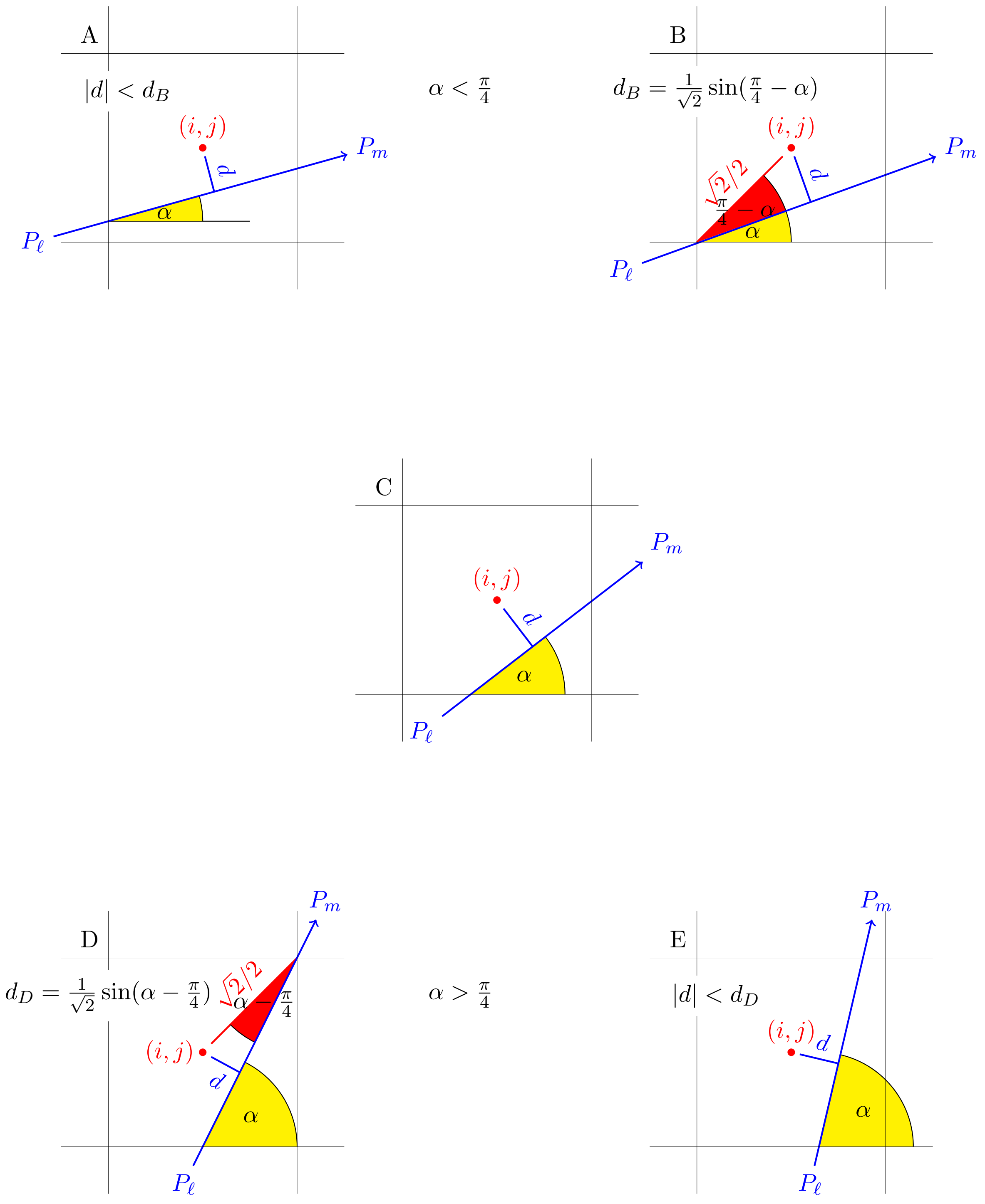
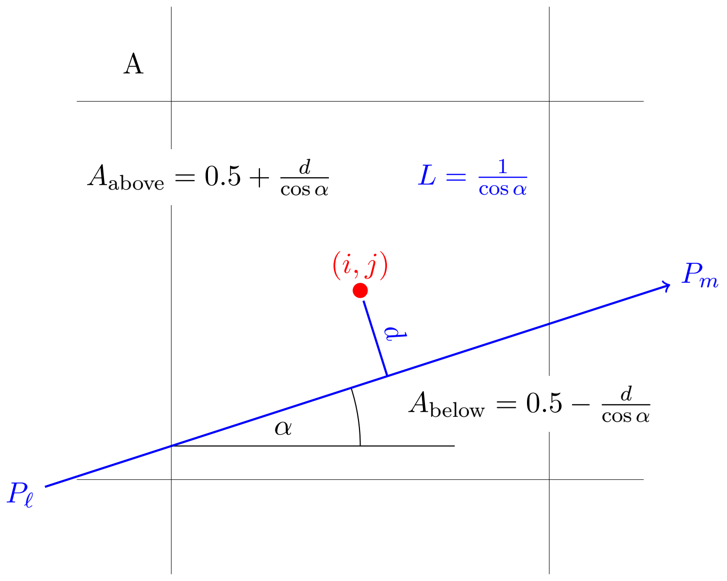


Figure 4: Five possibilities of a lines going through a pixel; angles above $\frac{\pi}{4}$ or with negative d can be dealt with by symmetry considerations. The critical angles are $\alpha_B = \frac{\pi}{4} - \arcsin(\frac{2d}{\sqrt{2}})$ and $\alpha_D = \arcsin(\frac{2d}{\sqrt{2}}) + \frac{\pi}{4}$. Figure adapted from [4].



A
$A_{\text{above}} = 0.5 + \frac{d}{\cos\alpha}$
$L = \frac{1}{\cos\alpha}$
(i, j)
P_m
d
$A_{\text{below}} = 0.5 - \frac{d}{\cos\alpha}$
α
P_ℓ

Figure 5: Length and Areas for case A. Note that this case is not possible for $|d| > 0.5$; additionally, $\sqrt{2}d < \sin(\frac{\pi}{4} + \alpha)$. Figure adapted from [4].

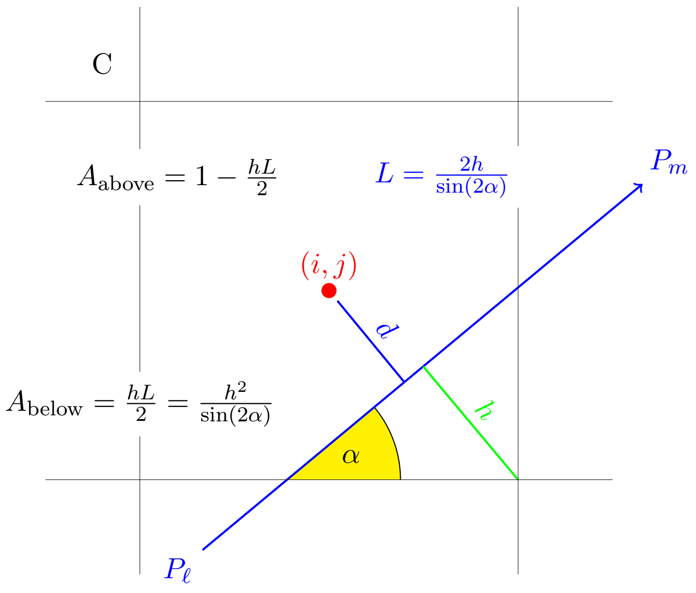

C
$A_{\text{above}} = 1 - \frac{hL}{2}$
$L = \frac{2h}{\sin(2\alpha)}$
P_m
(i, j)
d
h
$A_{\text{below}} = \frac{hL}{2} = \frac{h^2}{\sin(2\alpha)}$
α
P_ℓ

Figure 6: Length and Areas for case C. The calculations involve the height h of the triangle described by the line and a corner of the pixel. d and h must add up to the displacement necessary to move a line of angle α from the center to the corner, or $h = \frac{\cos(\pi/4-\alpha)}{\sqrt{2}} - d$. Figure adapted from [4]

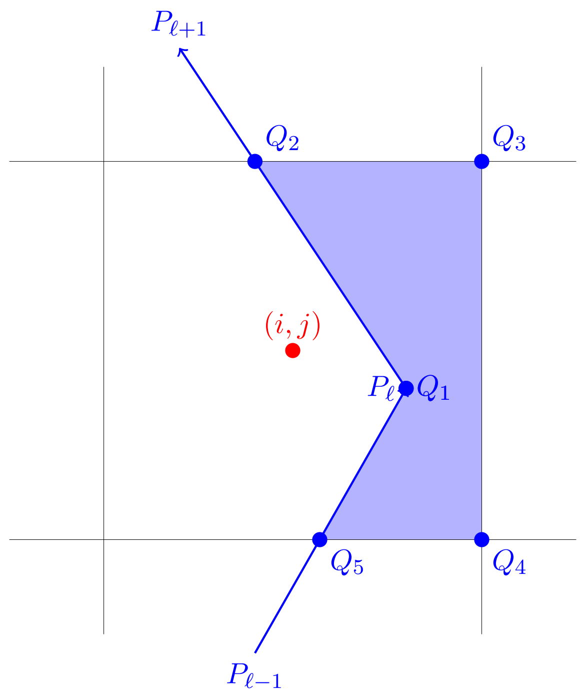
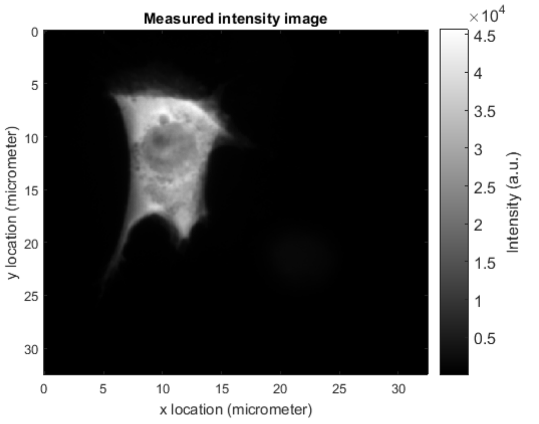
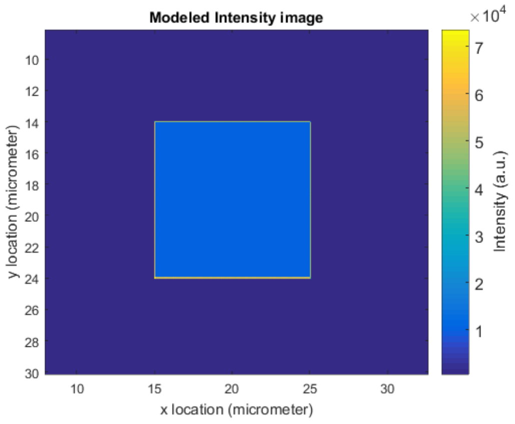


Figure 7: Calculation for a corner inside a pixel using a new Polygon Q. Figure adapted from [4]

To model an experiment from a typical setup as mentioned in section 2.4, a few factors from such a setup have to be taken into account in the simulation. First the sample, a biological cell, is modeled by a 10 by 10 square to describe its membrane, this can be seen in figure 8. A real cell ofcourse does not have an infinitely small width as an edge of a polygon does, to correct for this an gaussian image filter is applied to a simulated image of the polygon. It was tried to model this more properly by generating multiple polygons with different intensities around the main cell describing polygon, but this proved to be too resource intensive in performing the simulations. Another aspect of this is that the membrane in real world samples is about 10 nm thick, this is way below the resolution limit of light microscopy, therefore it was decided to let the previous gaussian filtering be applied such that it broadens the intensity of the membrane to the resolution limit calculated from the specific setup's specifications. From previous data akin to figure 3 a typical membrane intensity of approximately 15.000 counts per pixel in the perpendicular polarisation could be determined. For the outside this was about 500 counts, while the inside had a typical value of 10.000 counts. Typical anisotropy values for this data were 0.3 for the membrane, 0.26 for the inside and 0 outside the cell. In a real world setup, ofcourse signal noise is a problem. To simulate this to each simulated image both shot noise and electronic noise was added. To each area of the polygon(inside, outside, border) an intensity value was assigned. A comparison between a previously recorded image and a simulated image of a cell can be seen in figure 9. It has to be noted that the image does not seem noisy to the naked eye, this is because of the large intensities of the membrane making noise hard to detect on the scale. In the real experiment the fluorophore mCherry was used to label the samples. This leads to the usage of an excitation filter of 540-580 nm, an emission filter of 590-670 nm and a dichroic mirror for 585nm. The objective used was a 100x magnification, 1 Numerical Aperture oil objective. These parameters are thus included in the model. The most important ones to note are the numerical aperture of the objective and the emission wavelength of the fluorphore, as those influence the resolution limit of the setup.

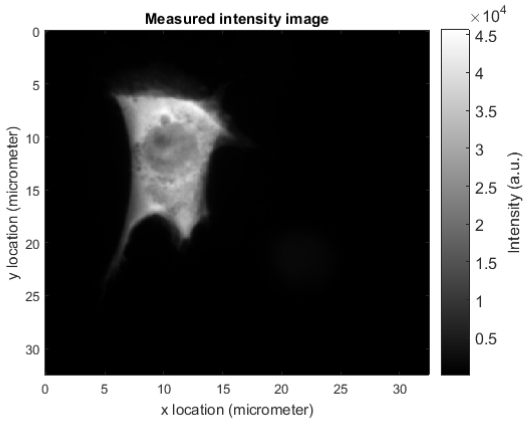


(a) Image recorded by Kyo Beyeler

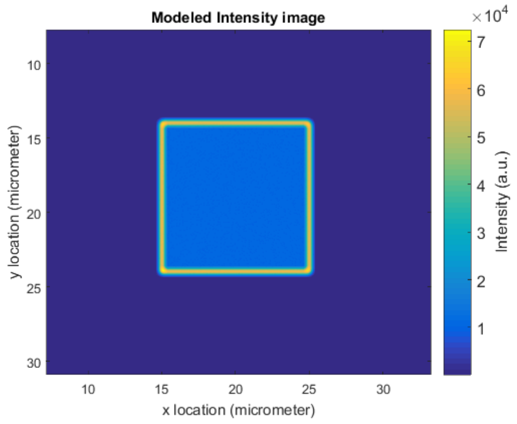


(b) Image generated by the model

Figure 8: Side by side comparison of an image of a cell recorded by the setup and an image generated by our model without any corrections.



(a) Image recorded by Kyo Beyeler



(b) Image generated by the model

Figure 9: Side by side comparison of an image of a cell recorded by the setup and an image generated by our model with corrections for membrane width.

3 Methods

Using our model of an image recorded through the experimental setup, images of both polarisations of a modeled cell can be made, and these can be used to calculate an anisotropy image using 1. In our simulation we could control what anistropy each component (inside,outside, membrane) of the cell would give. Using this as a baseline, the optimal results of the experiment could be set, by calculating the anisotropy from the parallel and perpendicular intensity images on a pixel by pixel basis. Then one of the two intensity images was purposely misaligned from the other with the misalignment found through previous measurements from the experimental setup.[6]

On these misaligned images a structure based method of anisotropy calculation was applied, while varying different parameters of the method as to characterize the method and find the optimal parameters.

3.1 Structure based analysis

The structure based method used is based on the fact that in real measurements the outside of the cell would have a very low intensity, the membrane has a very high intensity and the inside of the cell as an intensity somewhat in between these. If we would move along a line through a membrane we would thus expect a profile akin to figure 10.

Now if the two images are not properly registrated a misalignment of pixels occurs. The idea is to move along a line through the membrane and average the intensity over a pixel grid around the line and use these averaged values for the anisotropy calculation see figure 11. It is believed that this could be accurate enough to calculate the correct anisotropy within error margins while ofsetting the need for perfect image registration.

This would no longer calculate a full anisotropy image, but rather an anisotropy profile along the line through the membrane.

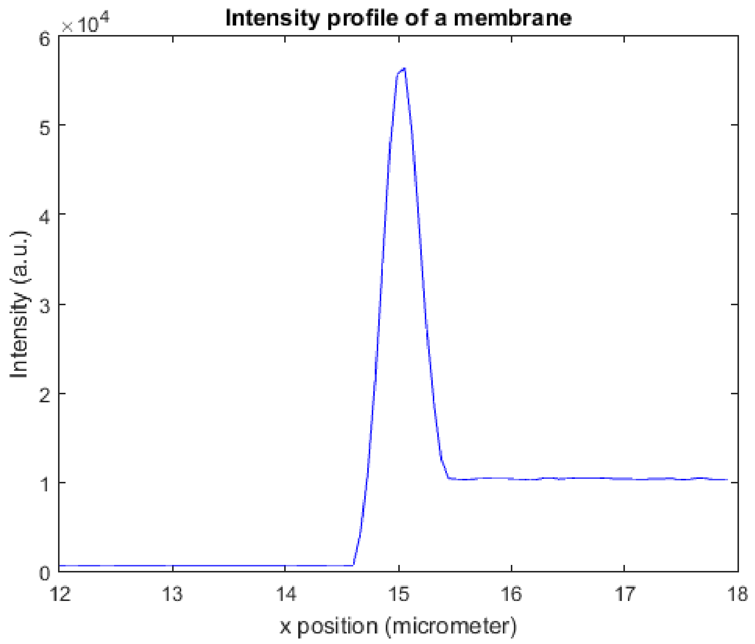


Figure 10: Expected intensity profile of the membrane of a cell. The outside having very low intensity, the membrane itself a very high intensity and the inside of the cell having an intensity somewhere in between.

To characterize this method, two parameters can be determined.

- The size of the pixel grid over which is averaged
- The amount of pixels moved along the line for each step

Pixel grids are characterised as (X, Y) where X is the number of pixels along the line and Y is the number of pixels perpendicular to the line. For example a (3,5) grid implies an averaging over 15 pixels, 3 pixels wide along the line and 5 pixels long perpendicular to the line. To determine an optimum for these parameters simulations were run and analysed using this method while varying one of the parameters. For each parameter twenty simulations of the experiment were run, for each simulation thus an anisotropy profile was created of the simulated cel. Since the maximum of the anisotropy was expected at the membrane, the maximum of each of these profiles was subtracted from the maximum of a profile which was taken from the anisotropy image as calculated from the simulated intensity images. This difference was averaged over the twenty simulations(see figure 12). Thus a measure for the deviation from the ideal value of the anisotropy was found as to characterise the error which would be induced in an anisotropy calculation using this structure based method as opposed to the pixel based method.

In table 1 a summary of all basic experimental parameters used in the simulations can be found.

Of note here is that in our simulation the G factor is a single number for the entire setup and not a different number for each pixel. Only one intensity per cell area is given, this is the intensity of the perpendicular image, the parallel image's intensity is calculated from the desired anisotropy using equation 1. A square cell was modeled as this is the easiest polygon to model. As emission

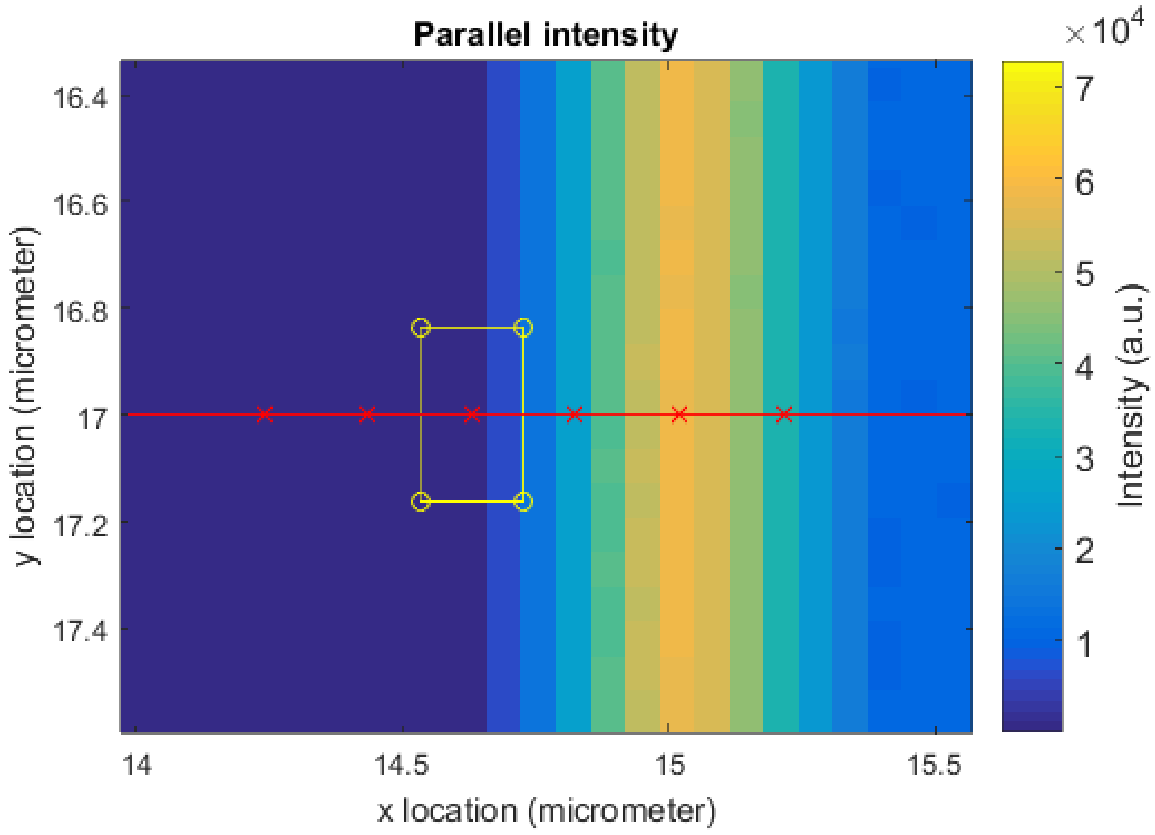


Figure 11: A representation of the structure based method applied on a parallel intensity image. The red line indicates the line along which is moved, with the red x-es indicating the center locations of subsequent grids. The yellow polygon indicates the grid of pixels over which is averaged.

parameter	value
inside intensity	5000 counts per pixel
outside intensity	500 counts per pixel
membrane intensity	15000 counts per pixellenght of membrane
image size	512x512 pixels
pixel size	6.5 μm x 6.5 μm
inside anisotropy	0.15
outside anisotropy	0
membrane anisotropy	0.3
cell	10 μm x 10 μm square cell
cell location(top left corner)	(14,15)
Numerical Aperture of the objective	1
magnification of the objective	100x
G factor	1.35
emission wavelength of fluorophor	610 nm

Table 1: Summary of all basic used experimental parameters

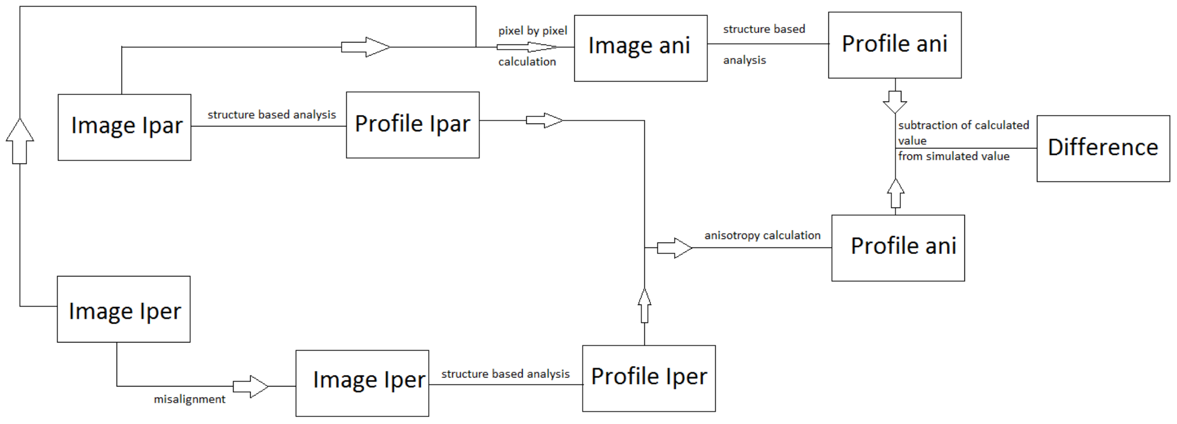


Figure 12: A shematic representation of the method used to characterize the error induced by the structure based analysis. From two simulated intensity images (one parallel and one perpendicular image) an anisotropy image is calculated. Then one intensity image is misaligned and from both images a profile is created. Then an anisotropy profile is calculated from both the anisotropy image and the intensity profiles. Finally the maxima of these profiles are subtracted to obtain the difference between them.

wavelength 610 nm was used as this is the emission wavelength of the fluorophor used in the actual experiment (mCherry).

4 Results

The first parameter of the structure based method which was investigated is the gridsize of the structure used in the analysis. Gridsize as a variable is noted as (X, Y). In this measurement X=3 pixels. The deviation from the simulated ideal value for different values of the gridsize along a straight line through the membrane can be seen in figure 13. The stability of this result is notable as it might have been expected that larger grids (and thus more averaging) would improve the result of the method. Remarkable is the fact that the deviation is always positive, thus always overestimating the anisotropy. The same measurements were made using a line which was not perpendicular to the membrane, but more diagonal, the results of this measurement can be seen in figure 14. There is a clear decrease in stability of the method for non perpendicular lines to be observed from this.

Second the influence of the amount of pixels moved between each step along the analysis was measured. This was done using a constant (3,3) gridsize. The result of this measurement can be seen in figure 15. Notable is the massive increase in deviation at stepsize 6 and 7 pixels. The lowering of the deviation at stepsizes 8 and 9 pixels is interesting as well, this might be explained through the idea that at these stepsizes one grid is localised more or less at the middle of the membrane. While at stepsizes 6 and 7 pixels this grid would be located away from the center of the membrane, thus causing a larger discrepancy with the expected value.

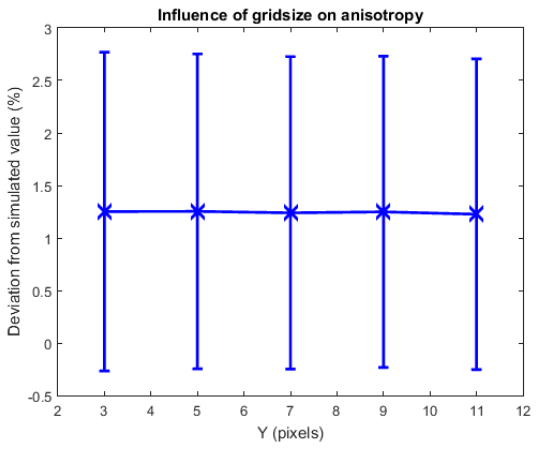


Figure 13: Plotted is the procentual deviation from the simulated value of the anisotropy of the membrane as a function of the size of the grid used for the structure based analysis method.

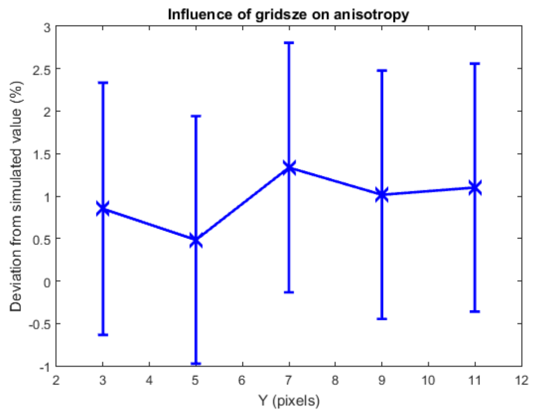


Figure 14: Plotted is the procentual deviation from the simulated value of the anisotropy of the membrane as a function of the size of the grid used for the structure based analysis method. This was done for a line nonperpendicular to the membrane.

Finally, the influence of the inner cell anisotropy on the membrane anisotropy was characterised, this result can be seen in figure 16. For this measurement a gridsize of (3,3) was used along with a stepsize of 3 pixels. It is interesting to see the structure based method seems to produce results which deviate less from the ideal value if the inner cell anisotropy is further away from the membrane anisotropy.

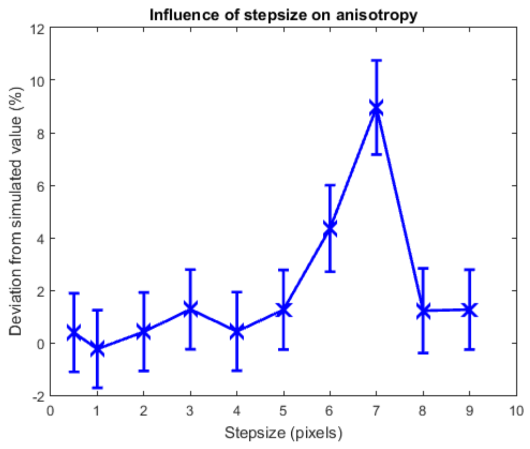


Figure 15: Plotted is the procentual deviation from the simulated value of the anisotropy of the membrane as a function of the amount of pixels moved along the line for the structure based analysis method.

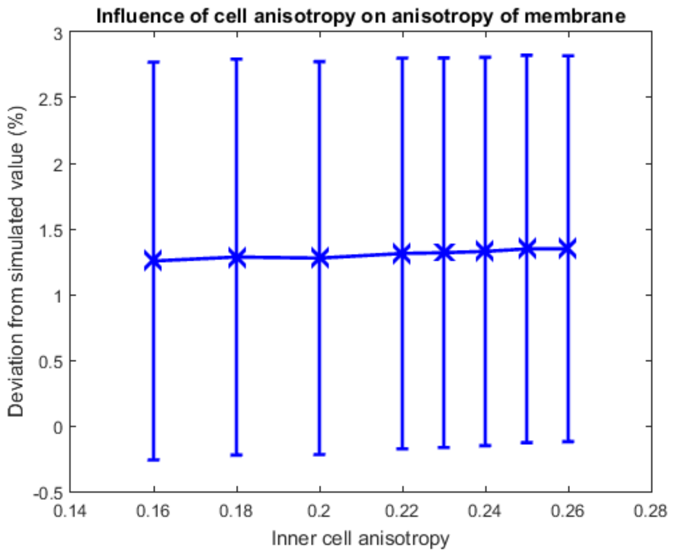


Figure 16: Plotted is the procentual deviation from the simulated value of the anisotropy of the membrane as a function of the inner anisotropy of the cell.

5 Conclusion and Discussion

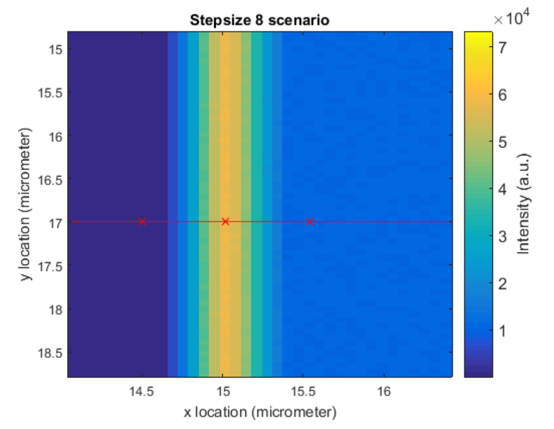
From the limited amount of measurements done to characterise our structure based method for anisotropy calculation, it seems to incur a deviation of $+1 \pm 1.5\%$ in the optimal case. Thus overestimating the actual anisotropy. This compares favorably to the standard deviation of earlier anisotropy measurements ($\approx 6\%$) [6], which do face problems with image registration. As the structure based method does not require perfect image registration, this result is highly promising. It shows that it might be possible to do anisotropy calculations without having perfect pixel to pixel image registration.

With regards to the size of the area averaged over in this structure based method the results seem counterintuitive as increasing the amount of pixels averaged over does not seem to influence the calculated anisotropy much. The fact that the standard deviation of the value does not decrease with gridsize is also remarkable. As it is expected that as we average over N pixels the standard deviation would decrease by a factor $\sqrt{N}$. This can be explained through the fact that the averaging occurs over the intensities which are then used to calculate the anisotropy through equation 1. In calculating the error in this calculation it turns out that the error in the intensity has only a minor influence on the final error in the anisotropy. The fact that this calculation has to be done thus offsets any gains in accuracy. It is shown that this result also holds for a line not perfectly perpendicular to the membrane of the cell although there is more fluctuation in the deviation in this case. Further research as to if the method does really hold for nonperpendicular lines is definitely required.

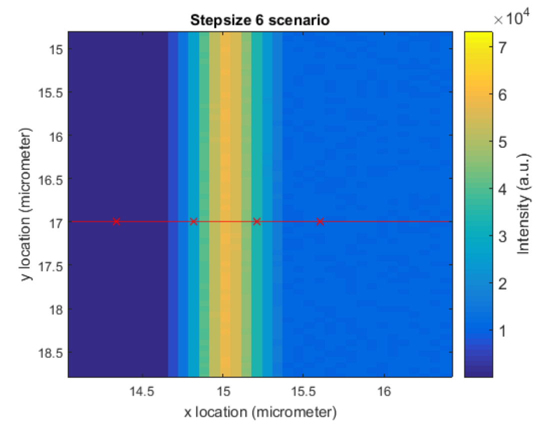
It has also been shown that it seems that oversampling(in our case 2 pixels stepsize and downward) does seem to lead to an increase in accuracy for the membrane anisotropy. As would be expected, undersampling (in our case from 4 pixels stepsize upwards) leads to a significant reduction in accuracy of the method, overestimating the anisotropy by up to approximately 9%. The reduction of deviation at 8 and 9 pixels stepsize can be accounted for by realising this happens due to the location of the grid being closer to the actual location of the membrane. This should be relatively easy to overcome, as a different choice of starting or end point of the line through the membrane changes the locations of the grid. It also shows it is important to make sure at least one point at which a grid is taken is located somewhat at the location of the membrane.

Finally the influence of the inner cell anisotropy is shown to be likely irrelevant as in the measured interval it showed very little to no influence on the measured anisotropy of the membrane. The variation in deviation from the simulated value does increase for values of the inner anisotropy closer to the membrane anisotropy, but only by 0.1%, well within the error margins. It has to be noted here that if the inner cell anisotropy does get too close to the value for the membrane anisotropy, the structure based method does break down. This is due to the fact that in this case the expected peak from the membrane is no longer clearly defined in the intensity profile generated by the structure based analysis.

The main points that require further attention would be the fact that only a limited number of variables of the method were checked together with the G-factor. Since the G-factor in the simulation is regarded as a single number for the entire setup, and not a per pixel number. Further research would incur improving the simulation by finding the exact per pixel G-factor of a setup



(a) The case of stepsize 8 pixels in our simulations.



(b) The case of stepsize 6 pixels in our simulations.

Figure 17: Comparison of different scenarios for the different stepsizes explaining the reduction in deviation at stepsizes 8 and 9 pixels. The red line indicates the line along which the structure based analysis is applied, with the x-es indicating the positions of the subsequent pixel grids. It is clear that at stepsize 6 the grids are located away from the membrane, thus calculating an less correct anisotropy than expected.

and taking this into account. Another point is that perfectly square cells were used, ofcourse real biological cells do not have this property, thus the method would have to be tested on measurements on actual biological samples and compared to other methods of anisotropy calculation to determine its viability. A final point of note is that the results of this thesis are calculated in a relative manner, thus not taking into consideration if the compared to simulated value might be lower than the actual entered value for the anisotropy. This could potentially mean that an increase of 1.5% in anisotropy actually means that the method is more accurate for misaligned images as opposed to interpreting the results as a deviation.

6 Acknowledgements

First of all massive thanks to Gerhard Blab for his supervision, guidance and help. You pushed me to look very carefully and detailed at all parts of the project thus aiding in fixing many errors in our simulation over time. Your humour and patience with me while explaining certain concepts helped a lot. I would furthermore like to thank everyone in the Molecular Biophysics group for being helpful when needed as well as just being an awesome bunch of persons with a great sense of humour and a great taste in cake, to all off you: Thanks for the wonderful time.

7 References

References

[1] Volkhard Helms, *"Fluorescence Resonance Energy Transfer"*, *Principles of Computational Cell Biology*, Weinheim: Wiley-VCH, p. 202, ISBN 978-3-527-31555-0, 2008.

[2] Gradinaru, Claudiu C.; Marushchak, Denys O.; Samim, Masood; Krull, Ulrich J., *"Fluorescence anisotropy: From single molecules to live cells"*, The Analyst 135 (3): 4529, 2010.

[3] J.R. Lakowicz, *Principles of Fluorescence Spectroscopy* (3rd ed., Springer, Chapter 10-12 deal with fluorescence polarization spectroscopy.), 2006.

[4] Gerhard A.Blab, *Generation of Test Patterns for Anisotropy Imaging*,Utrecht,2015.

[5] Paul Bourke, *Calculating The Area And Centroid Of A Polygon*,1988.

[6] Kyo Beyeler, *Detect clustering in cells using Homo-FRET*,thesis, Utrecht, 2015.

8 Appendix A Simulation code

This appendix contains all the Matlab code to used to generate the data used.

8.1 Generating and calling analysis of data:

Main function of the simulation, takes the relevant experimental parameters and calculates both intensity images. From those the anisotropy image is calculated. Then the perpendicular image is misaligned and the structure based method is applied. All data along with their errors are the output.

```
function [ dataR,datasigma,calcR,calcsigma,Ipar,Iper,R,sigmaR] =...
         doLots( x,y,Poly,T,Sin,G,lambda,xbegin,xend,dim,stepsize)
%doLots:Calculates the Intensity and anisotropy images and performs a
%structure based analysis on it.
%(x,y) the position of the pixels in the image
%Poly  a Nx2 matrix containing the vertex points of the Polygon
% Sin  a structure containing the input signals and anisotropys
% T    the affine transformation used for the image registration (for
%       simulation usually the misregistration), enter whatever non
%       affine2d object you want if you want no misalignment
% G    the G factor of the setup
% lambda  the wavelength of the used setup
%xbegin  the position of the starting point for the structure based
%        analysis
%xend    the position of the endpoint for the structure based analysis
%dim     the dimension of the structure used for the analysis (in pixels)

if nargin<10, help doLots; return; end
dx = x(2)-x(1); dy=y(2)-y(1);
if ( abs(dx-dy) > eps ), error('square pixels required!'); return; end
```

```


%calculation of the intensity and anisotropy images
[ Ipar,bgpar,sigmabgpar,Iper,bgper,sigmabgper,R,sigmaR ] = ...
    BlurImage( x,y,Poly,Sin,G,lambda );

 %misregistration of the image and transforming the begin and end
 %coordinates for the analysis accordingly
 if isa(T, 'affine2d');
Iper = imwarp(Iper, T,'OutputView',imref2d(size(Ipar)));
 coord=[xbegin,xend]/dx;
transformbegin=T.T'*[ coord(1);coord(2);1];
transformend= T.T'*[ coord(3);coord(4);1];
xbegintrans=[transformbegin(1),transformbegin(2)]*dx;
xendtrans=[transformend(1),transformend(2)]*dx;
 else
     xbegintrans=xbegin;
xendtrans=xend;
 end
%removing some nonsensical data (might happen)
R(sigmaR>0.4)=0;
sigmaR(sigmaR>0.4)=0;

%structure based analysis of anisotropy
dataR=analysestruct(x,y, xbegin,xend,dim ,R,stepsize,0);
datasigma=analysestruct(x,y, xbegin,xend,dim ,sigmaR,stepsize,1);

%backgroundsubtraction
Iparcor=Ipar-bgpar;
Ipercor=Iper-bgper;

%correcting negative intensities (impossible)
indicespar=Iparcor<0;
indicesper=Ipercor<0;
Iparcor(indicespar)=-Iparcor(indicespar);
Ipercor(indicesper)=-Ipercor(indicesper);

% cutoff on intensity to reduce noisy image of the outside (after bg
% subtraction, everything with less intensity than a basic outside pixel is
% defininitely outside or nonsense so gets set to 1 as to provide a clean
% image of the anisotropy, more specifically, definite outside now has an
% anisotropy of 0)
 Iparcor(Iparcor<Sin.I.Iout*dx^2)=1.35;
 Ipercor(Ipercor<Sin.I.Iout*dx^2)=1;


%structure based calculation of anisotropy
structIpar=analysestruct(x,y,xbegin,xend,dim ,Iparcor,stepsize,0);
structIper=analysestruct(x,y,xbegintrans,xendtrans,dim ,Ipercor,stepsize,0);

sigmaIpar=sqrt(Iparcor);
sigmaIper=sqrt(Ipercor);

structsigmaIpar=analysestruct(x,y,xbegin,xend,dim ,sigmaIpar,stepsize,1);
```

```
structsigmaIpar(:,3)=structsigmaIpar(:,3)/sqrt(dim(1)*dim(2));

structsigmaIper=analysestruct(x,y,xbegin,xend,dim ,sigmaIper,stepsize,1);
structsigmaIper(:,3)=structsigmaIper(:,3)/sqrt(dim(1)*dim(2));


calcR=(structIpar(:,3)-structIper(:,3).*G)./...
      (structIpar(:,3)+2.*structIper(:,3).*G);
calcsigma=sigma_sim(structIper(:,3),structsigmaIper(:,3),...
    structIpar(:,3),structsigmaIpar(:,3),G,0.05*G,...
    bgpar,bgper,sigmabgpar,sigmabgper);


%some plotting code to check the data

direction=xend-xbegin;

figure

%makes sure that if the line is vertical the plot still makes sense
if direction(1)<eps
errorbar(dataR(:,2),dataR(:,3),datasigma(:,3),'-b')
xlabel('y')
ylabel('anisotropy')
title('simulated anisotropy')
else
   errorbar(dataR(:,1),dataR(:,3),datasigma(:,3),'-b')
   xlabel('x')
   ylabel('anisotropy')
   title('simulated anisotropy')
end

figure

%makes sure that if the line is vertical the plot still makes sense
if direction(1)<eps
errorbar(dataR(:,2),calcR,calcsigma,'-b')
xlabel('y')
ylabel('anisotropy')
title('calculated anisotropy')
else
   errorbar(dataR(:,1),calcR,calcsigma,'-b')
   xlabel('x')
   ylabel('anisotropy')
   title('calculated anisotropy')
end


%plotting of intensity analysis, could be used for verification purposes,
%uncomment if desired
% figure
%
% %makes sure that if the line is vertical the plot still makes sense
% if direction(1)<eps
% errorbar(dataR(:,2),structIper(:,3),datasigma(:,3),'-b')
% xlabel('y')
% ylabel('anisotropy')
% title('Iper')
% else
%    errorbar(dataR(:,1),structIper(:,3),datasigma(:,3),'-b')
%    xlabel('x')
%    ylabel('anisotropy')
%    title('Iper')
% end
% figure
% if direction(1)<eps
% errorbar(dataR(:,2),structIpar(:,3),datasigma(:,3),'-b')
% xlabel('y')
% ylabel('anisotropy')
% title('Ipar')
% else
%    errorbar(dataR(:,1),structIpar(:,3),datasigma(:,3),'-b')
%    xlabel('x')
%    ylabel('anisotropy')
%    title('Ipar')
% end

end
```

8.2 Generation of Intensity and anisotropy data:

Function that generates both blurred intensity image, as well as the per pixel anisotropy image. Required input is all relevant parameters of the image desired.

```
function [ Itotalparnoise,bckgpar,sigmabckgpar,...
           Itotalpernoise,bckgper,sigmabckgper,...
           Rreconstructcorrected,sigmaR ] =...
           BlurImage( x,y,Poly,Sin,G,lambda )
%generates an anisotropy image using the entered anisotropy of the border,
%inside and outside of a cell as well as the intensity of the inside and
%outside of the cell (per micrometer^2) and the intensity of the cell
%border (per micrometer).All of this is contained in Sin, which is a
%structure containing two additional structures for I and R.
%The cell has a shape defined by a polygon Poly
%(this gives the vertex points of the cell),
%G is the G factor of the setup.
%x,y, give the center locations of the pixels (pixel 1 will be
%located at (x(1),y(1)) etc...)
%lambda is the used wavelength in micrometers

if nargin<6, help BlurImage; return; end
dx = x(2)-x(1); dy=y(2)-y(1);
if ( abs(dx-dy) > eps ), error('square pixels required!'); return; end

%getting the variables out of the Sin structure
Iinper=Sin.I.Iin;
```

```
Ioutper=Sin.I.Iout;
Iborderper=Sin.I.Iborder;
Rin=Sin.R.Rin;
Rout=Sin.R.Rout;
Rborder=Sin.R.Rborder;



M=length(x);
N=length(y);

[Min,Mout,Mborder]= polygonImage( x, y,Poly ); %calculating masks

%applying a gaussian filter to the masks to emulate the membrane not being
%a line and scales the intensities of the filtered border mask back to the
%entered intensity

NA=1; %Numerical Aperture of the setup
q= (0.61*lambda/NA)/dx; % resolution limit (FWHM of distribution) in pixels

sigma= sqrt((-(q/2)^2)/(2*log(0.5)));  %sigma of the distribution

%stores the maximum of the non filtered mask for scaling purposes
max1b=max(Mborder(:));
max1in=max(Min(:));
max1out=max(Mout(:));

%filtering to simulate resolution limit
Min=imgaussfilt(Min,sigma);
Mout=imgaussfilt(Mout,sigma);
Mborder=imgaussfilt(Mborder,sigma);

%maximum of the filtered mask
max2b= max(Mborder(:));
max2in=max(Min(:));
max2out=max(Mout(:));

%scaling factor
relativeb= max1b/max2b;
relativein=max1in/max2in;
relativeout=max1out/max2out;

%keep track of definite outside/inside
borderindices=Mborder==0;
inindices=Min==0;
outindices=Mout==0;

%actual scaling
Mborder=(Mborder-mean(Mborder(:)))*relativeb +mean(Mborder(:));
Min=(Min-mean(Min(:)))*relativein +mean(Min(:));
Mout=(Mout-mean(Mout(:)))*relativeout +mean(Mout(:));

%makes sure the scaling does not mess with definite outside/inside
Mborder(borderindices)=0;
```

```
Min(inindices)=0;
Mout(Mout<0)=0;
Mout(outindices)=0;

%factor between Iparallel and Iperpendicular (based on desired anisotropy
%values and G-factor) is used to calculate
% the desired parallel intensities
Iinpar=(-(G+2*G*Rin)/(Rin-1))*Iinper;
Ioutpar=(-(G+2*G*Rout)/(Rout-1))*Ioutper;
Iborderpar=(-(G+2*G*Rborder)/(Rborder-1))*Iborderper;

%calculating the intensities as measured by a camera (hence the rounding)
Itotalpar= round((Min*Iinpar+Mout*Ioutpar+Mborder*Iborderpar));
Itotalper= round((Min*Iinper+Mout*Ioutper+Mborder*Iborderper));

%adding shot noise
scale=1*10^12;
Itotalparshot=scale*imnoise(Itotalpar/scale,'poisson');
Itotalpershot=scale*imnoise(Itotalper/scale,'poisson');

Itotalparnoise=zeros(M,N);
Itotalpernoise=zeros(M,N);
% generates on average 5 counts of noise and adds them =electronic noise
for i=1:N
    for j=1:M
 Itotalparnoise(i,j)=Itotalparshot(i,j)+abs(5*GaussianNoise(1));
    end
end
for i=1:N
    for j=1:M
 Itotalpernoise(i,j)=Itotalpershot(i,j)+abs(5*GaussianNoise(1));
    end
end




%takes pixels which are over 99% outside for background calculation
 indices=Mout>0.99*dx^2;

%background reduction/calculation
bckgpar=mean(Itotalparnoise(indices));
bckgper=mean(Itotalpernoise(indices));

%number of pixels used for calculating the background
[number,~]=size(Mout(indices));
sigmabckgpar=bckgpar/sqrt(number);
sigmabckgper=bckgper/sqrt(number);

%background subtraction
correctedpar=Itotalparnoise-bckgpar;
correctedper=Itotalpernoise-bckgper;

%calculating the anisotropy
```

```
%Rreconstruct=(Itotalpar-G*Itotalper)./(Itotalpar+2*G*Itotalper);%no noise
%Rreconstructnoise=(Itotalparnoise-G*Itotalpernoise)./...
%              (Itotalparnoise+2*G*Itotalpernoise);%shot + electronic noise
Rreconstructcorrected=(correctedpar-G.*correctedper)./...
    (correctedpar+2*G.*correctedper); %noise+backgroundcorrection

%assuming 5% error in G for now
sigmaG=0.05*G;

%calculate the sigma of the anisotropy
sigmaR=sigma_sim(Itotalpernoise,sqrt(Itotalpernoise)...
    ,Itotalparnoise,sqrt(Itotalparnoise),G,sigmaG,...
    bckgpar,bckgper,sigmabckgpar,sigmabckgper);

end
```

8.3 Generation of masks for image generation:

Function that calculates the three masks (inside, outside, border)required to build an image of a polygon.

```
function [ Min, Mout, Mborder ] = polygonImage( x, y, Poly, level )
% function [ Min, Mout, Mborder ] = polygonImage( x, y, Poly )
% generate three matrices describing outside, inside, and border
% of a polygon Poly on a pixel grid.
% P is a 2xN matrix of vertex coordinates
% x and y are the center coordinates of the pixel grid

Min=[]; Mout=[]; Mborder=[];
if nargin<4, level = 1; end
if nargin<3, help polygonImage; return; end
if isempty(x) || isempty(y), error('need pixel grid data!'); return; end

if (level > 64), error('too many recursions! Abort!'); end


% make sure pixels are square
dx = x(2)-x(1); dy=y(2)-y(1);
%if ( abs(dx-dy) > eps ), error('square pixels required!'); return; end

NX = length(x); NY=length(y);

Min=zeros(NX,NY); Mout=Min; Mborder=Min;    % initialize
dL = dx/sqrt(2); % maximum distance from line that may be inside pixel
dA = dx^2; % area of a pixel

area = polygonArea(Poly);   % get orientation of polygon
if (area < 0)   % flip
    Poly = flipud(Poly);    % reverse order -> make (mathematically) positive
    fprintf(1,'flipped polygon to make it counter-clock-wise.\n');
end
P=[Poly; Poly(1,:)];  % make polygon cyclic (ie P0->P1->...->PN->P0)
```

```

%% define outside and find limits for loop
xmin = find (x < (min(Poly(:,1)-dL)));
if ~isempty(xmin), xstart = max(xmin)+1; else xstart =1; end
xmax = find (x > (max(Poly(:,1)+dL)));   xend = min(xmax)-1;
if ~isempty(xmax), xend = min(xmax)-1; else xend = NX; end
ymin = find (y < (min(Poly(:,2)-dL)));   ystart = max(ymin)+1;
if ~isempty(ymin), ystart = max(ymin)+1; else ystart =1; end
ymax = find (y > (max(Poly(:,2)+dL)));   yend = min(ymax)-1;
if ~isempty(ymax), yend = min(ymax)-1; else yend = NY; end

% mark outside as outside
Mout([xmin(:);xmax(:)],:)=1; Mout(:,[ymin(:);ymax(:)])=1;

% calculate border lines in the form ax + by + c = 0
NP = size(Poly,1);

L = [ P(1:NP,2)-P(2:end,2) P(2:end,1)-P(1:NP,1)]; SQ = sqrt([diag(L*L'),...
            diag(L*L'),diag(L*L')]);
    % a and b, normalization sqrt(a^2+b^2)
L = [L, -P(1:NP,2).*L(:,2)-P(1:NP,1).*L(:,1)]./SQ;
% reuse the factors already calculated; c = -(y0 a + x0 b)



alpha = atan2(L(:,2),L(:,1));
for ii = xstart:xend
    for jj = ystart:yend
        distances = L * [x(ii); y(jj); 1]; % calculate distance pixel -> lines

       pxl_inside = all(distances > dL);   % def -> inside is positive L
       pxl_outside = any(distances < -dL);

        if (pxl_outside)
            Mout(ii,jj) = 1;
        elseif (pxl_inside)
            Min(ii,jj) = 1;
        else
            % now for the interesting part
            NL_idx = find(abs(distances)<dL);   % how many lines are close?
            NL = length(NL_idx);
            switch (NL)
                case 0 % this should not happen
                    warning('ERROR. Divide by cucumber. Please reinstall universe!');
                    Mborder(ii,jj) = -1;
                case 1 % default - we are close to the line
                    [pxl_in, pxl_out, pxl_border] = ...
                        linePixel(distances(NL_idx)./dx, alpha(NL_idx));
                    Min(ii,jj) = pxl_in;
                    Mout(ii,jj) = pxl_out;
                    Mborder(ii,jj) = pxl_border;
                otherwise % corner, or other complications
                    Pts_idx = pointsPixel(Poly, [x(ii), y(jj)], dx/2);
                    NPts = length(Pts_idx);
                    if (NPts == 1) && (NL == 2) %% a corner!
                        [pxl_in, pxl_out, pxl_border] =  ...
                            cornerPixel(Poly(mod([-2 -1 0]+Pts_idx,NP)+1,:), ...
                                [x(ii), y(jj)], dx);
                        Min(ii,jj) = pxl_in;
                        Mout(ii,jj) = pxl_out;
                        Mborder(ii,jj) = pxl_border;
                    else % time to supersample!
                        %fprintf(1,'supersample at (%.2f, %.2f): NL = %d,...
                        %NPts = %d.\n',...
                        %    x(ii), y(jj), NL, NPts);
                        [pxl_in, pxl_out, pxl_border] = polygonImage( ...
                            x(ii)+dx*[-0.25, 0.25], y(jj)+dx*[-0.25, 0.25], ...
                            Poly, level+1);
                        Min(ii,jj) = sum(pxl_in(:));
                        Mout(ii,jj) = sum(pxl_out(:));
                        Mborder(ii,jj) = sum(pxl_border(:));
                    end % switch
            end % switch
        end % inside - outside - border
    end % for jj
end % for ii

if level<2 %scales the masks to the pixel size
Min = Min*dA; Mout = Mout*dA; Mborder = Mborder*dx;
else % if level >2 then we have been supersampling, since supersampling
     % divides a pixel into 2x2 pixels the area of a superssampled pixel is
     % 0.25 the original pixel and the pixel length is 0.5 the original
     % pixel so we scale by those amounts if we have supersampled
    Min=Min*0.25; Mout=Mout*0.25; Mborder=Mborder*0.5;
end

%% EOF
```

8.4 Calculation for area of a polygon:

Subfunction that calculates the area of a polygon

```
function area = polygonArea(Poly)
% function area = polygonArea(Poly)
% return the area of the polygon described by Poly;
% the area may be negative, depending on the orientation of the polygon
% (clockwise = mathematically negative)

area = 0;
if (nargin<1) || (isempty(Poly)) || size(Poly,2)<2
  return;
end

N = size(Poly,1);
P = [Poly; Poly(1,:)];

```

```
% replace sum with matrix multiplication
area = 0.5*(P(1:N,1)'*P(2:end,2) - P(2:end,1)'*P(1:N,2));

%% EOF
```

8.5 Calculation for a line through a pixel:

Subfunction that handles the calculation of amount of inside, outside and border contained within a pixel for the case a line moves through the pixel.

```
function [ pxl_in, pxl_out, pxl_border ] = linePixel( d, angle )
% function [ pxl_in, pxl_out, pxl_border ] = linePixel( d, angle )

pxl_in=0; pxl_out=0; pxl_border=0;

if (nargin<2) || isempty(d) || isempty(angle), return; end

invert = (d<0); % in this case we need to swap inside and outside later

halfPi = pi/2; quartPi = pi/4;
alpha = mod(angle, halfPi);
d = abs(d);

if (alpha < eps) && (abs(d-0.5) < eps)
    % this line goes along the edge of the pixel; 0 and -pi/2 are part
    % of the pixel (left and bottom, thus (i-0.5/j-0.5)

    % if d>0, the 'allowed' angles are -pi/2 and 0
    inside_pixel =  (angle > -3*quartPi) && (angle < quartPi);
     % avoid asking exact values - shorter than abs(angle - pi/2) < eps & ...

    if xor(invert, inside_pixel)
        % we belong
        pxl_border = 1;
        pxl_out = invert;
        pxl_in = 1-pxl_out;
    else
        % we do not belong
        pxl_border = 0;
        pxl_in = invert;
        pxl_out = 1-pxl_in;
    end % do we belong?
    return; % we are done!
end



if (alpha > quartPi), alpha = halfPi-alpha; end

if (d < sin(quartPi+alpha)/sqrt(2)) % inside the pixel!
    dcritical = sin(quartPi-alpha)/sqrt(2);
    if (d > dcritical) % case C
```

```
        h = cos(quartPi-alpha)/sqrt(2)-d;
        pxl_border = 2*h/sin(2*alpha);
        pxl_out = h*pxl_border/2;
        pxl_in = 1-pxl_out;
    else % case A
        pxl_border = 1/cos(alpha);
        dArea = d*pxl_border;
        pxl_in = 0.5 + dArea;
        pxl_out = 0.5 - dArea;
    end
else
    pxl_border = 0; % line outside pixel
    pxl_in = 1; pxl_out = 0;
end


if (invert) % need to switch inside and outside
    % pxl_border is the same
    placeholder = pxl_in;
    pxl_in = pxl_out;
    pxl_out = placeholder;
end % if invert

%% EOF
```

8.6 Finding points of the Polygon in a pixel:

Subfunction that finds the points of the entered polygon located inside a pixel.

```
function point_idx = pointsPixel(Poly, P, delta)
% function point_idx = pointsPixel(Poly, P)
% find points of the polygon in pixel around point P,
% delta is the half-size of the pixel
%
% note: we also find points on the border with the adjacent pixels to avoid
% a pathological situation in the decision tree (point in corner)

xP = Poly(:,1); yP = Poly(:,2);
point_idx = find( (xP >= P(1)-delta) & (xP <= P(1)+delta) & ...
    (yP >= P(2)-delta) & (yP <= P(2)+delta) );

%% EOF
```

8.7 Calculation for a corner in a pixel:

Subfunction that handles the calculation of amount of inside, outside and border contained within a pixel for the case a corner of the polygon is located inside the pixel.

```
function [ pxl_in, pxl_out, pxl_border ] = cornerPixel( Poly, P, dx )
% function [ pxl_in, pxl_out, pxl_border ] = cornerPixel( Poly, P, dx )
% Poly contains the three points generating the corner, only one of which is
% in the pixel; P is the position of the pixel center, dx the size of the pixel

pxl_in=0; pxl_out=0; pxl_border=0;

if (nargin<2) || isempty(Poly) || isempty(P), return; end
if (nargin<3) || isempty(dx), dx=1; end

pxlBorder=[0.5 0.5; -0.5 0.5; -0.5 -0.5; 0.5 -0.5]; % pixel borders
P0=(Poly(2,:)-P)/dx; % pivot point in pixel
P1=(Poly(1,:)-P)/dx;
P2=(Poly(3,:)-P)/dx;
dP1 = P0-P1; dP1 = dP1/norm(dP1);    % [a1 b1]
dP2 = P2-P0; dP2 = dP2/norm(dP2);    % [a2 b2]
Lp = polygon2Lines(pxlBorder);
numerator = Lp*[P0'; 1];
denominator1 = Lp*[dP1'; 0];
denominator2 = Lp*[dP2'; 0];
lambda1 = numerator./denominator1;
lambda1 = lambda1(lambda1>0);
lambda2 = -numerator./denominator2;

%catching the case in which the point is located on a pixel border
 if all(lambda1<0)
 lambda1=0;
else
    lambda1 = lambda1(lambda1>=0);
end

if all(lambda2<0)
 lambda2=0;
else
  lambda2 = lambda2(lambda2>=0);
end
lambda1_smallest = min(lambda1);
lambda2_smallest = min(lambda2);
P1 = P0 - lambda1_smallest*dP1;
P2 = P0 + lambda2_smallest*dP2;

phi1 = atan2(P1(2), P1(1)); % minus x(i) and y(j) ...
phi2 = atan2(P2(2), P2(1));
%phi0 = atan2(P0(2), P0(1));

Q=P0;
if (max(abs(P2))- 0.5<eps) % inside; P1-[x(i) y(j)]
  Q=[Q;P2];
  pxl_border = norm(P2-P0);
end

% TODO: double check that I move the right way
% ie consistent with the defintion of inside/outside the main polygon!

piFourths = pi/4;
corners = [3 1 -1 -3 3]*piFourths; % look for first corner
```

```
%catching the case if phi2 is smaller than -3/4 pi
if phi2<= -3/4*pi
    phi2=phi2+2*pi;
end

phi = corners(corners<phi2);

 if(isempty(phi))
     phi=(-3*piFourths);
 end
phi=phi(1);
if (phi1>phi2) phi1 = phi1 - 2*pi; end

while phi > phi1
  Q = [Q; [cos(phi) sin(phi)]/sqrt(2)];
  phi = phi - pi/2;
end



if (max(abs(P1))-0.5<eps)  % inside; P1-[x(i) y(j)]
  Q = [Q; P1];
  pxl_border = pxl_border + norm(P1-P0);
end

pxl_in = polygonArea(Q);
if pxl_in<0, pxl_in = pxl_in + 1; end
pxl_out = 1-pxl_in;

%% EOF
```

8.8 Calculation for two line through a pixel:

Subfunction that calculates the coëfficients of lines describing the border of a polygon.

```
function L = polygon2Lines(Poly)

L = [];
if (nargin<1) || (isempty(Poly)) || size(Poly,2)<2
  return;
end

N = size(Poly,1);
P = [Poly; Poly(1,:)];

L = [ P(1:N,2)-P(2:end,2) P(2:end,1)-P(1:N,1)];
SQ = sqrt([diag(L*L'),diag(L*L'),diag(L*L')]); % a and b, normalization sqrt(a^2+b^2)

L = [L, -P(1:N,2).*L(:,2)-P(1:N,1).*L(:,1)]./SQ;
% reuse the factors already calculated; c = -(y0 a + x0 b)

%% EOF
```

8.9 Generation of Gaussian noise:

Function that generates random gaussian noise with standard deviation sigma.

```
function [ output ] = GaussianNoise(sigma)
%Generates a random deviation from 0 for the standard deviation sigma


      u1 = rand(1);
      u2 = rand(1);


   z0 = sqrt(-2.0 * log(u1)) * cos(2*pi * u2);
   z1 = sqrt(-2.0 * log(u1)) * sin(2*pi * u2);
   output= z0 * sigma ;

end
```

8.10 Calculation of sigmas from the simulation:

Subfunction that calculates all the sigmas of a simulation.

```
function [ sig_ani ] = sigma_sim( Iper,sigmaIper,Ipar,sigmaIpar,G,sigmaG,bgpar,bgper,...
                                  sigmabgpar,sigmabgper)
%calculatess the sigma of the anisotropy image from everything that goes in
%Derivatives used here are calculated with Mathematica.

a=1.69;
G = double(G);
Ipar = double(Ipar);
Iper = double(Iper);


%---derivatives and uncertainties------------------------------
denom = (  bgpar+2.*bgper.*G -Ipar -2.*G.*Iper).^2;%denominator

d_ani_I_par = double( ( 3.*G.*(Iper-bgper))./denom ); %derivative
sig_I_par = double(sigmaIpar); %uncertainty

d_ani_I_per = double( (3.*G.*(-Ipar+bgpar))./denom );%derivative
sig_I_per = double(sigmaIper);%uncertainty

d_ani_bgI_par = double( (3.*G.*(Iper+bgper))./denom );%derivative
sig_bgI_par = sigmabgpar;%uncertainty

d_ani_bgI_per = double( (3.*G.*(Ipar-bgpar))./denom );%derivative
sig_bgI_per = sigmabgper;%uncertainty


d_ani_G = double( (-3.*(bgpar-Ipar).*(bgper-Iper))./denom );%derivative
sig_G = double(sigmaG);%uncertainty

```

```

%-----------------------------------------------------------

sig_ani = ((             (d_ani_I_par.*sig_I_par).^2 ...
                       + (d_ani_I_per.*sig_I_per).^2 ...
                       + (d_ani_bgI_par.*sig_bgI_par).^2 ...
                       + (d_ani_bgI_per.*sig_bgI_per).^2 ...
                       + (d_ani_G.*sig_G).^2 ...
                        ).^(1/2));

%if silly unphysical things happen, set them to 0
sig_ani(isnan(sig_ani))=0;
sig_ani(isinf(sig_ani))=0;

end
```

9 Appendix B analysis code

This appendix contains all the Matlab code used to analyse data generated by the simulation.

9.1 Structure based analysis:

Function that handles the structure based analysis. Outputs a list of positions and averaged values. Takes as input the locations of the pixels, begin and endpoint for the analysis, gridsize, the image to analyse, the stepsize and a logical which should be 1 if the image to analyse is an uncertainty image (which changes the way the average should be taken).

```
function [ imageout ] = analysestruct(x,y,xbegin,xend,dim ,imagein,stepsize,logical)
%analyses a grid pixels of size dim along a line from xbegin to xend of
%imagein, note for dim that dim(2) is perpendicular to the line and dim(1)
%is along the line so a 3x5 grid will be 5 pixellenghts perpendicular and 3
%pixellenghts along the line
%(x,y) give the coordinates of the pixels
%logical is a logical that indicates if the imagein is a uncertainty image
%or not (changes the way in which the values are calculated)

if nargin<7, help analysestruct; return; end
dx = x(2)-x(1); dy=y(2)-y(1);
if ( abs(dx-dy) > eps ), error('square pixels required!'); return; end

dx=x(2)-x(1);%lenght of pixel
direction=xend-xbegin; %direction of the line along which we analyze the pixels
directionnorm=direction/norm(direction); %normalised direction of the line

%number of steps needed to go along the line
steps=(direction/directionnorm)/(dx*stepsize);

imageout=[];
position=[0,0];%initialisation position
for i=1:steps
```

```

    position=position+directionnorm*dx*stepsize;

    if i==1 %make sure the first iteration uses the starting position
        position=xbegin;
    end

     %calculates the Polygon to be used
    Polygon=calcPoly(dx,position,directionnorm,dim);
%plotting the Polygon on top of the image, uncomment if verification of
%this is needed.
%     figure
%     imagesc(x,y,imagein)
%     hold on
%     plot(Polygon(:,1),Polygon(:,2),'-bo')
%     plot([xbegin(1),xend(1)],[xbegin(2),xend(2)],'-rx')


    Min=polygonImage(x,y,Polygon); %determines which pixels to use
    Min=Min';
    %selects all inside pixels an calculates their weight factors
    number=Min(Min>0);
    sum1= sum(number(:));
    weights=number/sum1;
    sumw=sum(weights(:));


     %check if weights are correctly calculated uses eps *100
     %cause due to rounding errors it tends to go above eps
     %and E-14 is acceptable

    if abs(sumw-1)>eps*100
         fprintf(1,'position:%f and sumweights:%f',position(1),sumw)
         error('weights are wrong')
    end
    % if there are no pixels over half inside skip this step
    if ~isempty(number)
    %calculates a weighted average over all pixels over half inside the Polygon
    %if the image is an uncertainty image,
    %switch to the proper propagation method
         if logical
             val=sum((double(imagein(Min>0)).^2).*weights.^2);
             val=sqrt(val);
         else
             val=sum(double(imagein(Min>0)).*weights);
         end
         imageout=[imageout; [position,val]];
    end

end %for
end
```

9.2 Calculation of a Polygon for structure based analysis method:

Subfunction that calculates a polygon based on location of the pixel, direction of te line moved along in the structure based analysis and the desired size of the pixel grid for the analysis.

```
function [ Poly] = calcPoly( dx,location,direction,dim )
%Calculates the vertexes of a Polygon around location, direction is the
%direction along which we move the Polygon for structure based analysis,
%dim is the size of the structure (in pixels) which we want to analyse
%dx is the length of a pixel side



%finds the direction perpendicular to the movement
directionperp=[direction(2),-direction(1)];
%scales the normalised direction to the pixel size
directionnorm=direction/norm(direction)*dx;
%scales the normalised perpendicular direction to the pixel size
directionperpnorm=directionperp/norm(directionperp)*dx;

%calculate the four verteces
x1=location+1/2*dim(1)*directionnorm+1/2*dim(2)*directionperpnorm;
x2=location+1/2*dim(1)*directionnorm-1/2*dim(2)*directionperpnorm;
x3=location-1/2*dim(1)*directionnorm-1/2*dim(2)*directionperpnorm;
x4=location-1/2*dim(1)*directionnorm+1/2*dim(2)*directionperpnorm;

%put them in a Polynomial
Poly=[x1;x2;x3;x4];

end
```

9.3 Analysis of stepsize data:

Function used to analyse and plot stepsize data. Input is the path of the data.

```
function [ output_args ] = stepsize(path )
%stepsize: analyses the data in path for stepsize influence and plots it

stepsizes=[0.5,1,2,3,4,5,6,7,8,9];
differences=[];
sigmas=[];

[difference,sigma]=averaging( strcat(path,'\3_3_0.5'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\3_3_1'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\3_3_2'));
differences=[differences;difference];
sigmas=[sigmas;sigma];
```

```

[difference,sigma]=averaging( strcat(path,'\3_3_3'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\3_3_4'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\3_3_5'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\3_3_6'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\3_3_7'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\3_3_8'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\3_3_9'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

figure
errorbar(stepsizes,differences,sigmas,'-bx','linewidth',2, 'MarkerSize',12)

end
```

9.4 Analysis of gridsize data:

Function used to analyse and plot gridsize data. Input is the path of the data.

```
function [ output_args ] = gridsize( path )
%gridsize:analyses the data in path for gridsize influence and plots it

stepsizes=[3,5,7,9,11];
differences=[];
sigmas=[];

[difference,sigma]=averaging( strcat(path,'\3x3'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\3x5'));
differences=[differences;difference];
sigmas=[sigmas;sigma];
```

```

[difference,sigma]=averaging( strcat(path,'\3x7'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\3x9'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\3x11'));
differences=[differences;difference];
sigmas=[sigmas;sigma];


figure
errorbar(stepsizes,differences,sigmas,'-bx','linewidth',2, 'MarkerSize',12)
end
```

9.5 Analysis of inner anisotropy data:

Function used to analyse and plot inner anisotropy data. Input is the path of the data.

```
function [ output_args ] = innerani(path )
%stepsize: analyses the data in path for inner anisotropy influence and
%plots it

stepsizes=[0.16,0.18,0.20,0.22,0.23,0.24,0.25,0.26];
differences=[];
sigmas=[];

[difference,sigma]=averaging( strcat(path,'\0.16'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\0.18'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\0.20'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\0.22'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\0.23'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\0.24'));
differences=[differences;difference];
```

```
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\0.25'));
differences=[differences;difference];
sigmas=[sigmas;sigma];

[difference,sigma]=averaging( strcat(path,'\0.26'));
differences=[differences;difference];
sigmas=[sigmas;sigma];


figure
errorbar(stepsizes,differences,sigmas,'-bx','linewidth',2, 'MarkerSize',12)

end
```

9.6 Averaging of multiple datasets:

Subfunction used to calculate the average value and sigma of 20 datasets.

```
function [ verschilend, sigmaend ] = averaging( path)
%averaging: averages the differencess between simulation and calculation
%and their errors for twenty datasets located at path and outputs them

verschiltotaal=[];
sigmatotaal=[];

[verschil,sigma]=verwerking( strcat(path,'\data1' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data2' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data3' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data4' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data5' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data6' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data7' ));
```

```
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data8' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data9' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data10' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data11' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data12' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data13' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data14' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data15' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data16' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data17' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data18' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data19' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];

[verschil,sigma]=verwerking( strcat(path,'\data20' ));
verschiltotaal=[verschiltotaal;verschil];
sigmatotaal=[sigmatotaal,sigma];
```

```
verschilend=mean(verschiltotaal); % taking the mean of all the differences
sigmaend=mean(sigmatotaal)/sqrt(20); %taking the mean of all the sigmas



end
```

9.7 Finding the deviation and its sigma of a dataset:

Subfunction that finds a maximum in the simulated anisotropy data and a maximum in the anisotropy data from the structure based analysis. Determines the difference between the two and its sigma.

```
function [ verschil, sigma ] = verwerking( filename )
%verwerking: finds maximum in a simulated dataset and maximum in the
%calculated data (should also be a maximum if everything is ok)
%and then finds the corresponding sigmas, subtracts the calculated value
%of the maximum from the simulated value and determines its sigma.

%loads the file and the relevant variables containing the anisotropy data
 file=matfile(filename);
 calcR=file.calcR;
 calcsigma=file.calcsigma;
 dataR=file.dataR;
 datasigma=file.datasigma;

%finds the maxima
 maxR=max(dataR(:,3));
 maxRcalc=max(calcR(:));
%finds the corresponding sigmas
 sigma1=datasigma(abs(dataR-maxR)<eps);
 sigma2=calcsigma(abs(calcR-maxRcalc)<eps);
%determines the difference between simulation and calculation
%(in procentual representation) and its sigma
 verschil=100*(maxRcalc-maxR)/maxR;
 sigma=sqrt((sigma1^2)*((100/maxR)^2) +...
     (sigma2^2)*((100*maxRcalc/(maxR^2))^2));




end
```