CHALMERS
UNIVERSITY OF TECHNOLOGY

MasterThesis/figure/Segmentation.png

Image Analysis of Yeast Cell Lineages

Image Analysis of Yeast Cell Lineages Using Microscopy time-lapse Data

KLAS HOLMGREN

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026

Master's thesis 2026

KLAS HOLMGREN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Cvijovic Lab
Chalmers University of Technology
Gothenburg, Sweden 2026

Image Analysis of Yeast Cell Lineages
Image Analysis of Yeast Cell Lineages Using Microscopy Video Data
KLAS HOLMGREN

Supervisor: Barbara Schnitzer, Department of Mathematical Sciences
Examiner: Marija Cvijovic, Department of Mathematical Sciences

Master Thesis 2026
Department of Mathematical Sciences
Applied Mathematics and Statistics
Cvijovic Lab
Chalmers University of Technology

Cover: Segmentation of yeast cells in microscopy image.

Typeset in LaTeX
Gothenburg, Sweden 2026

Image Analysis of Yeast Cell Lineages
KLAS HOLMGREN
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

The aim of this thesis was to extract cell lineages from microscopy images displaying a growing yeast cell population. To do so, a program has been developed that is able to detect cells, track them over successive images and collect physical information of the cells, such as cell sizes, positions and mother-daughter relationships in the lineage.

Several thresholding methods have been implemented and tested to detect the cells in the images. Thresholding was used to segment the edges of the cells and a convex hull has been applied to segment the cell area. Further, the Watershed algorithm was investigated but did not perform well due to the variability in cell sizes and the lack of good parameter values. Edge detection methods, such as the Laplacian Gaussian or the Canny method, could detect the edges, however didn't perform well when the cells were closely packed to each other. Lastly, a random forest model was trained using manually annotated data resulting in similar results as the thresholding methods.

In the next step, cell sizes were extracted as the area of the segmentation and positions as the centroid of the segmentation. To connect mother to daughter cells in the lineage, the cell positions and the activity of a protein called Whi5 that indicates cell growth were used. Incorporating these two parameters, a measure to estimate the suitability of each mother-daughter connection was defined and used to reconstruct the complete lineage from the microscopy data.

Finally, limitations and potential improvements of the data and the analysis are discussed. The extracted data can in the future provide new insights into aging, by for example validating existing mathematical models with data.

Keywords: Yeast, Image Analysis, Ageing, *Whi5*, Thresholding, Watershed algorithm, Laplacian, Gaussian, fluorescence.

Acknowledgements

This Project was conducted at CvijovicLab located in the Department of Mathematical Sciences at Chalmers University of Technology.
Thanks to Barbara Schnitzer for supervising the project. Thanks to Marija Cvijovic for being the Examiner for the project. Thanks to Stina Karlsson for proof reading of the project.

Klas Holmgren, Gothenburg, September 2020

Contents

List of Figures

1
Introduction

In systems biology, model validation is the crucial part in building, constructing and simulating mechanistically correct models. However, obtaining and analysing appropriate experimental data is non-trivial. One of the common techniques for data generation is microscopy. This generates images of given biological phenomena that are consequently analyzed using different image analysis techniques.

To better understand ageing in unicellular organism, Schnitzer et al [22] developed an dynamic model of replicative ageing. The model describes an aging yeast population, predicting age-dependent conditions for cell division, and the effect of aging on cell-growth. To make experimental verification of the models microscopy data have been collected of budding yeast. Under the microscope, cells with different ages of the founder cell are observed during budding. To indicated cell division, a specific fluorescent marker has been added to the cells. This is also captured in the microscopy through a fluorescent channel. This reports builds on fluorescent time-laps microscopy data collected by Irene Delgado Román a guest PhD student in CvijovicLab and a PhD student in Sebastián Chávez group at the Instituto de Biomedicina de Sevilla, Sevilla, Spain.

Image analysis is a common tool in biological studies where video or image is used as a documentation tool. The growing generation of image-based data poses a great methodological challenge in image processing and analysis [7]. There are number of available tools for extracting data from microscopy images. Such as ImageJ[14] as well as programs building upon this like Fiji[12], TrackMate[16] and MaMuT [17]. However, these programs are older, Java-based and not trivial to develop. In this report a new Python-based tool was developed using novel methods. This brings researchers an alternative to already existing tools. Typically, available tools use methods from the field of image analysis. Image analysis is the process in which one tries to extract useful information from digital images. This is done by using different image processing techniques.

The first step is usually image compression which aims at removing the redundancies in an image. This step could include for example cropping of the image. After compression, it is common to apply image preprocessing. Here one might want to take the image gradient to find edges [6], smooth it with a Gaussian filter to reduce noise [9], or use thresholding to isolate objects based on color. Lastly, one tries to analyze the image with available algorithms such as edge detectors, blob detection, or trying segmentation with a Watershed Algorithm [19]. This final step will result

in a classification of the image such that each pixel in the image is belonging to a certain object or no object.

Methods used in image analysis generalize well to videos, as well, since a video only consists of an array of images. But a series of additional possibilities apply to videos. One of which is tracking an object through time. This is called tracking and consists of multiple steps. First is linking, this is the process of linking one detection of an object in one frame to the same object in the next frame. Next one has to fill the gaps where the object avoids detection and finally one has to filter noise detections. Several different algorithms for noise detectiopn exist, such as Optical Flow [3], Mean-shift[8] and centroid tracking[15]. Regardless of the algorithm used, as a final output, a trace of detections tracking the object through the video is produced.

In this report several methods are investigated with the aim to track cell lineages. All previously mentioned steps, preprocessing, segmentation and tracking have been implemented, resulting in the fully constructed lineage giving explicit mother daughter relationships.

2

Theory

2.1 Noise reduction using Gaussian and median filters

In order to understand how Gaussian and median filter works the concept of convolution need to be covered. Convolution is a common concept used in image analysis. Convolution provides a way of multiplying two matrices that can be of different sizes. In our case we want to multiply our image with a smaller Kernel.

Mathematically one could write the process of convolution as

$$\hat{I}(x,y) = \sum_{k=1}^{m}\sum_{l=1}^{n} I(x+k-1, y+l-1)K(k,l) \tag{2.1}$$

Here x runs from 1 to $M-m+1$ and y runs from 1 to $N-n+1$. The output image $\hat{I}(x,y)$ is the convoluted image with the kernel applied to every pixel.

How the edges of the image is handled differ in each implementation.

Images coming from microscopy can be noisy that is why it some times can be useful to do some noise reduction. There are many different noise reduction techniques. But most of them build on the idea of combining a pixel with nearby pixels in an intelligent way.

One of these ways are called Gaussian filtering. Here a Gaussian kernel is used to smoothen the image (Figure 2.1). This can however result in loss of information so should be applied with caution.

The Kernal looks as following

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

note that this kernel is normalized thus not changing the intensity of the image.

Figure 2.1: Example of Gaussian Kernel of size one (left) and example of a Gaussian kernel of size 3 (right).

Another denoising method is called *median filtering*. This is a simple noise filter that takes the mean of the nearest pixels. The kernel is simply a uniform distribution.
A related method *Non-linear mean* uses a non linear kernel to take the weighted mean for the nearby pixels. The pixels are weighted according to a weight function $f(x)$ using the pixels as parameter.
One function that could be used as kernel f is wavelet-based estimator of the noise standard deviation. This will return a greater value if the pixel at $I(x, y)$ is closer to $I(\hat{x}, \hat{y})$ and a lower value if they are not similar. Thus only the pixels with similar intensity are merged.
The function assumes that the noise is Gaussian distributed. And the estimate is taken from the median absolute deviation of the wavelet detail coefficients. Described in section 4.2 in [10].
Considering that images are often not of the same size, creates the need to re-scale the images. To expand an image one can use either pixel replication or interpolation. This is different ways of filling the empty pixels when scaling an image.

2.2 Increasing contrast using histogram equalization

Histogram equalization is a method for image enhancement without losing any information. It works by increasing the contrast in the image.
The algorithm looks like following.
Let $p(i)$ be the proportion of the image with intensity i.The accumulative distribution then becomes

$$cdf_I(i) = \sum_{j=0}^{i} p(j). \tag{2.2}$$

To equalize the histogram one would like a linear accumulative distribution on the

form

$$cdf_{\hat{I}}(i) = iK. \tag{2.3}$$

This is the transform that gives the most separation in intensity without any loss of information. However there are algorithms resulting in less histogram equalization such as *Contrast Stretching* and *Adaptive equalization* as illustrated in figure 2.2. They work in the same way looking at the accumulative disruption and applying a transform to a certain function.

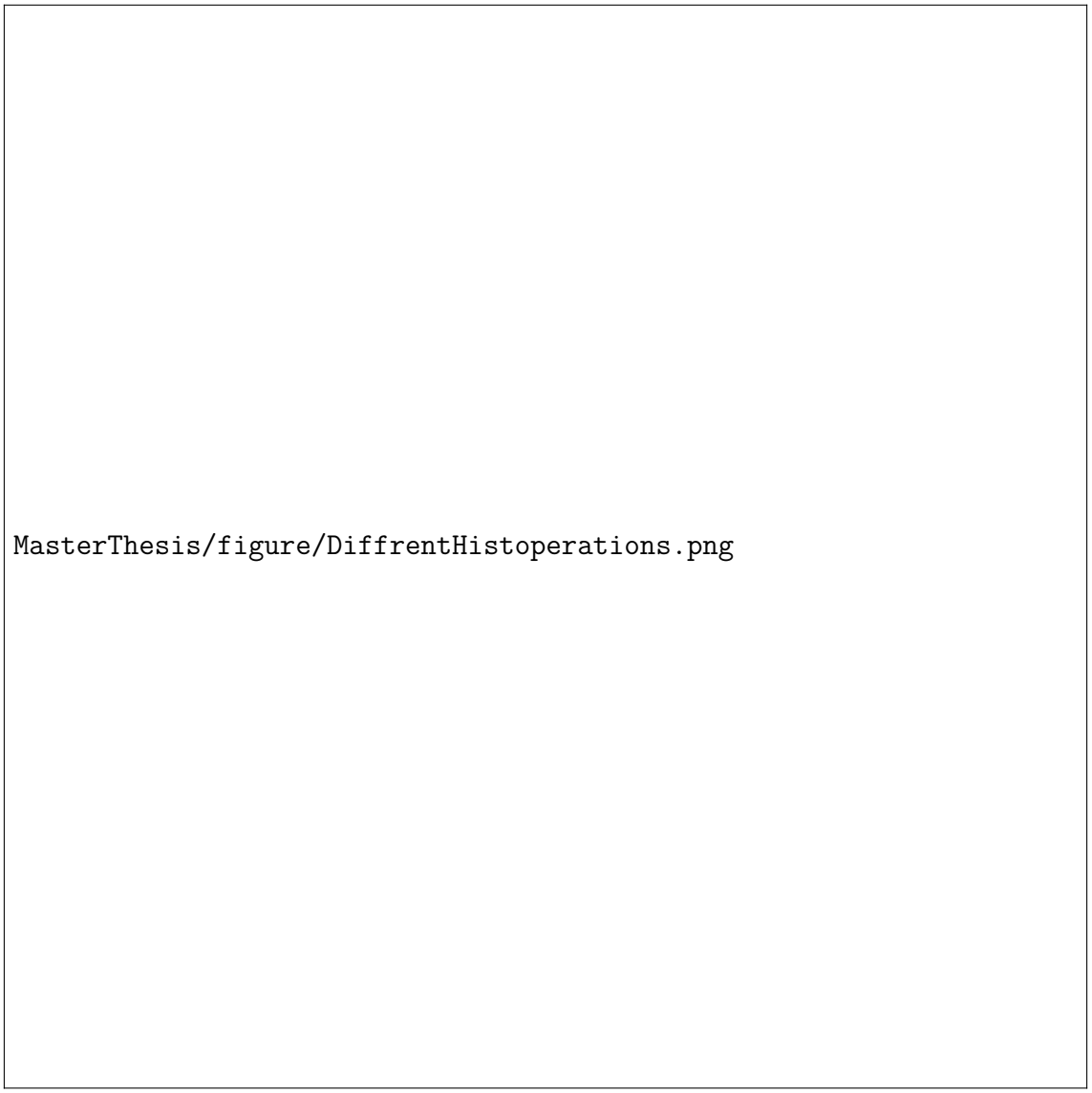

Figure 2.2: Illustration of different histogram operations. Notice that all of them work by stretching the histogram using different transforms [1].

2.3 Algorithms for segmentation

The goal of segmentation is to identify sub-images that represents objects. The segmentation or partitioning can be exhaustive meaning that every part of the image

$I(x, y)$ is covered by the regions R_i with $i \in [1, N]$

$$\cup_{i=1}^{N} R_i = I(x, y). \tag{2.4}$$

A partitioning can also be exclusive meaning that no partition have a common region with any other region.

$$R_i \cap R_j = \emptyset, \quad i \neq j. \tag{2.5}$$

There are several available algorithms to achieve this segmentation. In the case of yeast cell segmentation every pixel will be labeled as either cell or not cell and cells will not be able to overlap. Meaning that our segmentation will be both exhaustive and exclusive. In this section the theory behind the algorithms explored in this work is presented.

2.3.1 Thresholding with manual threshold and Otsu-binarization

Thresholding is one of the simplest methods for segmenting an image. It simply creates a mask for pixels with an intensity lower or higher than a certain threshold. In colored images with multiple channels this is called multiband thresholding. This method works when there is a clear distinction between foreground and background. The transformed binary image $\hat{I}$ can be written as

$$\hat{I}(x, y) = \begin{cases} 1 & , I(x, y) \in T \\ 0 & , I(x, y) \notin T \end{cases} \tag{2.6}$$

with T as the threshold region. $\hat{I}$ is then a binary image.
There are multiple ways of finding a good threshold value. A common method is setting it manually by locking at the image. One could also set it automatically by for example Otsu-binarization which minimizes the variance in each class created by the threshold. The intra-class variance can be written as a weighted sum of the variances of the two classes.

$$\sigma_{\omega}^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t) \tag{2.7}$$

The weights are obtained from proportion of the image covered by a certain class. These proportions can be found by summing the proportion of the image consisting of intensity i. With the proportion represented by $p(i)$ this gives

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i) \tag{2.8}$$

and

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i). \tag{2.9}$$

This threshold value can be found by doing a exhaustive search over all possible threshold to then returning the one with the least variance.

Figure 2.3: Example of Otzu Binarization on a image from the fluorescent channel. To minimize the variance in each class the threshold is set to separate background and foreground.

Thresholding is also a first step in many more complicated segmentation algorithms such as the Watershed Algorithm.
Another useful concept is the convex hull. This is the set of all convex combinations of a set of points. A convex combination of a set of points is a linear combination of these points where all combinations are non negative and sum to 1. Intuitively one could think of this like filling a shape. This is useful in segmentation problems where the objects in question are convex.

2.3.2 Watershed algorithm

The Watershed Algorithm is another classic method for segmentation. It can be very useful when objects are touching. It works by finding the center of the object then expanding outwards like a watershed.
The first step usually involves some kind of thresholding resulting in a binary image with foreground and background. This is called the opening threshold. After thresholding a distance transform is applied. This results in a gray-scale image where every pixel in the foreground is colored according to its distance to the boundary or background. The result is an image where pixels near the center of the objects are given a high value and pixels closer to the edge a lover value. The distance transform $\hat{I}$ of the image I can be written as

$$\hat{I}(x,y) = \min_{I(x,y)=1, I(x_0,y_0)=0} (\sqrt{(x-x_0)^2 + (y-y_0)^2}). \tag{2.10}$$

Another threshold is applied to the distance transform this is done to find the centers of each object i.e. the highest values in the distance transform. This will give the number of objects in the image as well as there center positions. To find which pixels belong to which object the threshold is increased until a region from one object intersects with a region from another object. Where two object intersect is

marked as the separation between the objects and the boundary of the objects have been identified [4].

2.3.3 Edge detection algorithms Laplacian Gaussian and Canny method

Another way of detecting objects are by looking at the change in intensity. This is called edge detection. One simple way of finding edges is by taking the Laplacian of the image. However this method is sensitive to noise in the image and therefor it is important to first apply a Gaussian filter. After this has been done the Laplacian can be applied without detecting small instances of noise.
The Laplacian $\mathcal{L}(x, y)$ of an image are taken with respect to the x and y positions in the image. With I as the intensity of the image the Laplacian can be written as:

$$\mathcal{L}(x, y) = \frac{\partial^2 I}{\partial^2 x} + \frac{\partial^2 I}{\partial^2 y}. \tag{2.11}$$

Since the image is a discrete function the Laplacian have to be approximated and there are different methods to do so. The pixels can be given values according to the absolute value of the Laplacian:

$$L = \sqrt{L_x^2 + L_y^2} \tag{2.12}$$

After this has been applied the edges should be clearly distinguishable and easy to find. The entire object can be segmented by segmenting the convex hull of the detected edges [18].
Another method for edge detection is the *Canny method*. This method also starts out with applying a Gaussian filter for noise reduction. To the edges or the intensity gradient of the image an edge operator is used. There are different ways of approximate the gradient such as Roberts, Perwit, Sobel but all of them returns the gradient in horizontal G_x and vertical axis G_y.
Then the arctangens with two parameters are taken. This gives the angel of the vector from the horizontal axis

$$\Theta = arctan(G_x, G_y)$$

The angel Θ is then rounded to either the horizontal axis 0 deg, vertical 90 deg or one of two diagonals 45 deg or 135 deg. This gives one image with the intensity and one with the angel of the image.
To increase the visibility of the edges, a non-maximum suppression is applied. This increases the contrast by setting values except the local maximma to zero. It works by comparing the current pixel to the one in the gradient and negative gradient direction calculated previously. If the current pixel is the largest it is kept otherwise it is suppressed i.e. set to zero. Then the edges are filtered with two threshold values. The once above the highest are classified as strong edges always saved, the ones above the lower threshold but bellow the highest is suppressed or saved depending on context and the ones bellow the lowest threshold are always suppressed [6].

2.3.4 Machine learning algorithm random forest

Machine learning algorithms are algorithms that learn through experience. Its common to use a training data set and a testing data set. The algorithm builds a model through the use of the training set. The model is then tested using the test set. Algorithms in this category include linear regression, support vector machines (SVM), diction trees and Random Forest. In the case of segmentation the data is an image and the result is an image with classification.

Random forest consist of a large number of decision trees. A decision tree is a tree where it in every branch splits the data along the parameter that gives the largest separations between the classes. The random forest simply uses many decision trees to create a consensus vote for the classification of the data. In the case of image classification the pixels from the different channels can be used as variables. Also adding filtered images is possible to get more parameters for the model.

2.3.5 Deep learning segmentation with U-Net

Convolutional neural networks (CNNs) have become the dominant approach in image analysis tasks such as classification, object detection and segmentation. A CNN learns hierarchical feature representations directly from data by stacking layers of learnable convolutional filters. Each convolutional layer applies a set of filters to the input feature maps, producing new feature maps that capture increasingly abstract patterns with depth.

For semantic segmentation — where every pixel is assigned a class label — the network must produce an output of the same spatial resolution as the input. A common architectural pattern for this is the *encoder–decoder*. The encoder progressively reduces spatial resolution while increasing the number of feature channels, building a compact representation of the image. The decoder then upsamples this representation back to the original resolution. However, downsampling in the encoder discards spatial detail that is important for precise boundary localisation.

The *U-Net* architecture, introduced by Ronneberger et al. [21], addresses this by adding *skip connections* that concatenate feature maps from each encoder stage directly to the corresponding decoder stage (Figure 2.4). This allows the decoder to combine high-level semantic information from the bottleneck with fine-grained spatial information from the encoder, resulting in sharp, accurate segmentation boundaries. The architecture was originally proposed for biomedical image segmentation and has since become the standard baseline for cell segmentation tasks.

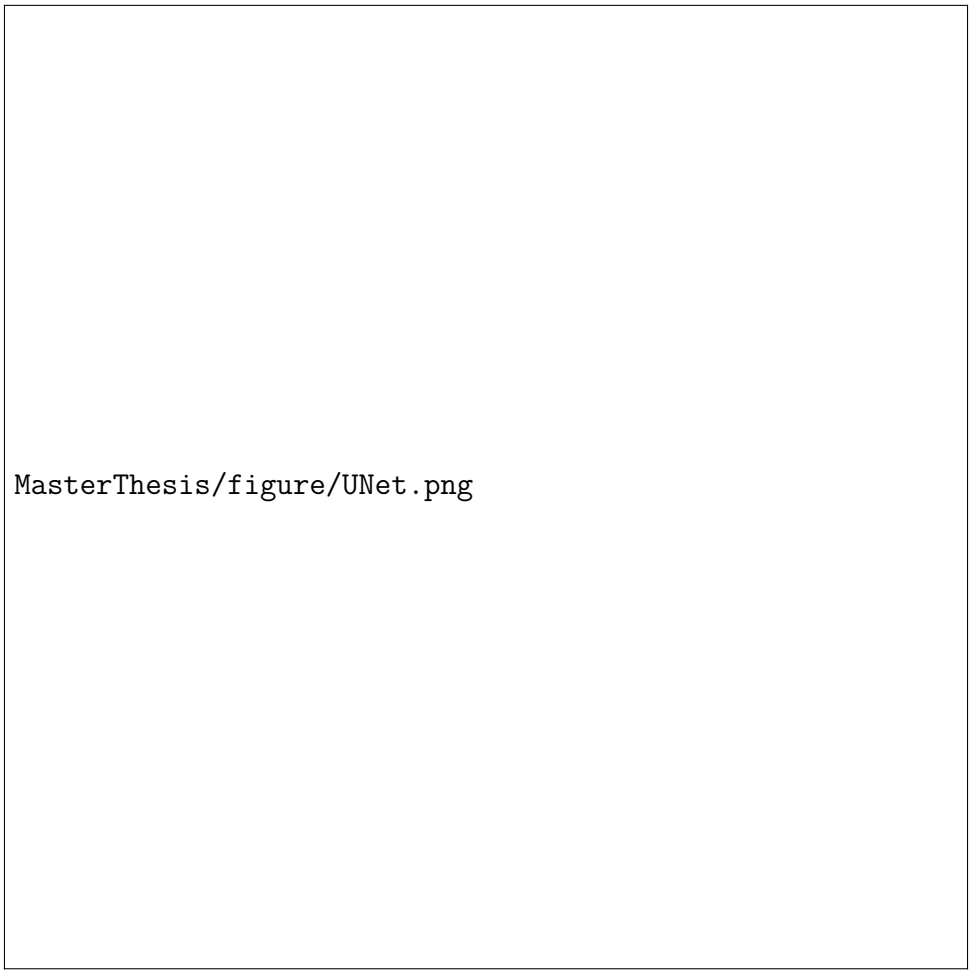

Figure 2.4: The U-Net architecture. The encoder (left) reduces spatial resolution through repeated convolution and max-pooling. Skip connections (horizontal arrows) carry feature maps directly to the decoder (right), which restores the original resolution through transposed convolutions. The final layer produces a per-pixel probability map.

The U-Net used in this work consists of four encoder blocks, each containing two 3×3 convolutional layers followed by batch normalisation and ReLU activation, and a 2×2 max-pooling operation. A bottleneck block connects encoder and decoder. The four decoder blocks each apply a 2×2 transposed convolution to upsample the feature maps, concatenate the corresponding skip connection, and apply two 3×3 convolutions. The final layer is a 1×1 convolution followed by a sigmoid activation, producing a single-channel probability map where values close to 1 indicate cell pixels.

2.3.5.1 Loss functions for binary segmentation

Training a binary segmentation network requires a loss function that measures the discrepancy between the predicted mask $\hat{M}$ and the ground-truth mask M. Two loss functions are commonly used.
Binary cross-entropy (BCE) treats each pixel independently and is defined as

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}\left[M_i \log \hat{M}_i + (1 - M_i)\log(1 - \hat{M}_i)\right] \tag{2.13}$$

where N is the total number of pixels, $M_i \in \{0, 1\}$ is the ground-truth label and $\hat{M}_i \in (0, 1)$ is the predicted probability for pixel i.
Dice loss measures the overlap between prediction and ground truth and handles class imbalance well, since cell pixels are typically a small fraction of the total image area. It is defined as

$$\mathcal{L}_{\text{Dice}} = 1 - \frac{2\sum_i \hat{M}_i M_i + \epsilon}{\sum_i \hat{M}_i + \sum_i M_i + \epsilon} \tag{2.14}$$

where ϵ is a small constant for numerical stability. The combined loss used for training is

$$\mathcal{L} = \mathcal{L}_{\text{BCE}} + \mathcal{L}_{\text{Dice}}. \tag{2.15}$$

2.4 Centroid tracking

Tracking is the process of following an object through a video. This can be done in several different ways but the end result is a knowledge of the position of the object throughout the video.
Linking is the process of pairing an object in one frame to the same object in the next frame. This can be done in several ways. One of which is centroid tracking. This method relies on the euclidean distance. It works by assigning the object in the next frame to the objects nearest in the previous frame. After the assignment is done the remaining objects are registered as new objects. If an object does not get an assigned a new instance in the next frame it is marked as disappeared until a new detection is assigned to the object. If it has been been marked as disappeared for a sufficiently long time it is removed from tracking.

3

Methods

3.1 Exploration of optical and fluorescent video data of budding yeast cells

The available data consists of two channels one is an optical channel and the other one is a fluorescent channel as seen in figure 3.1. Both channels have 4 focus levels each. The data set has been obtained by Irene Delgado Román a guest PhD student in CvijovicLab and a PhD student in Sebastián Chávez group at the Instituto de Biomedicina de Sevilla, Sevilla, Spain. The data set is composed of several time-lapse microscopy data of different ages of the founder cell.

After cell division, both mother and daughter secretes a protein called *Whi5* which can be detected through the fluorescent channel. This protein is expressed when the cells enter a new cell cycle and start growing after cell division. After division, both mother and daughter will light up simultaneously because they are both entering the growing phase. This can be used to connect the mother cell to the daughter cell thus tracking the lineages of the cells.

Examples from the channels are displayed below. In the optical channel foreground and background share similar colors but the edges are clear and are clearly distinguished in the fluorescent channel. However, the cells are closely packed and it is hard to distinguish the separation. One can also see the fluorescent light from the cells in the right corner indicating a cell division.

Figure 3.1: An example of a optical image (left) and an example of a fluorescent image (rigft).

The data comes in form of *Leica Image File Format* with the ending *.lif*. To be able to convert these files the package *Bioformats* was used [11]. A loading method was created using *Bioformats* converting a *.lif* file to *.tif* file. This can then be loaded to *Python* using OpenCV [5].

3.1.1 Image preprocessing with de-noising filters and histogram equalization

A couple of different image processing techniques were investigated. Among them were Gaussian filtering, median filtering and non-linear median filtering with sigma estimator. The images are also quite small so they were re-scaled with 10 times the size in both height and width to make them easier to analyze. After the filters were applied a method for histogram equalization was added to increase the contrast in the image resulting in a noise-reduced image with clear contrast.

3.2 Implementation of Segmentation Methods

In this section the specific method for implementing the segmentation is presented.

3.2.1 Thresholding manually and using Otzu binarization

The images were first prepossessed using a median filter with sigma estimator. A histogram was then plotted for every zoom-in level in the optical channel. This histogram was segmented with appropriate threshold values. Thresholding was also done with Otzu Binarization. This was able to capture the edge of the cells. Finally, the convex hull was applied to segment the middle of the cells and all connected components where identified and registered as detected and are linked together. Filtering has been applied to detect invalid detections that are returned for tracking.

3.2.2 Process of Watershed algorithm

The first step in the algorithm is finding a good opening threshold. Several thresholds were tried, like an Otzu threshold of the fluorescent channel and the Otzu threshold with a convex hull from the previous subsection. A distance transform was then applied to the threshold image creating a distance map illustrating the distance of each pixel in the foreground to the background. The center of the object was then detected using a threshold at 65% of the maximum value in the distance map. Lastly, a Watershed algorithm was used to classify the remaining areas from the threshold.

3.2.3 Edge detectors Laplacian Gaussian and Canny method

First, a Gaussian filter was applied to the optical image to reduce noise. Then the Laplacian was applied This resulted in an image with clearer edges. To further define the location of the edges the Canny edge detection was used. After this implementation, the method was not investigated further.

3.2.4 Machine learning algorithm random forest with manually annotated images

The random forest algorithm was implemented for the segmentation of the cells. The optical images as well as fluorescent images were used as input parameters as well as several filter images including median filter, variance filter, histogram equalization filters.
Seven images were annotated manually for training and two images for testing. This was done with binary encoding using Gimp. Several filters were applied to the images. Every filtered image was saved as a parameter in a data frame and the pixels were added as observations. Then a Random Forest model was trained using the annotated images as training data [13].

3.2.5 Deep learning segmentation using U-Net

A U-Net was implemented as an alternative segmentation method to complement the classical approaches described above. The goal is to learn a per-pixel cell/background classifier directly from the brightfield frames, producing a binary segmentation mask that can replace the Otsu binarization step in the pipeline.

3.2.5.1 Training data generation with Otsu pseudo-labels

Manually annotating every frame of a time-lapse microscopy video is labour intensive. Instead, a weakly supervised approach was adopted: the existing Otsu binarization pipeline (with morphological closing post-processing) was used to automatically generate binary mask pseudo-labels for each brightfield frame. Although these pseudo-labels are imperfect — inheriting the limitations of thresholding — they provide sufficient supervision to train a U-Net that generalises beyond what

thresholding alone can achieve, in particular for frames with uneven illumination or high cell density.
All frames were extracted from the optical channel video and resized to 256×256 pixels for training. The dataset was split into 85% training and 15% validation.

3.2.5.2 Data augmentation

To improve generalisation and compensate for the limited number of training frames, online data augmentation was applied during training. Each sample was randomly flipped horizontally and vertically with probability 0.5, and rotated by a uniformly sampled angle in $[-15\check{r}, 15\check{r}]$. The same transformation was applied to both the image and the mask to preserve their correspondence.

3.2.5.3 Network architecture and training procedure

The U-Net architecture described in Section 2.3.5 was implemented in PyTorch [20] with feature map sizes of 32, 64, 128, and 256 channels in the four encoder stages. The network was trained end-to-end by minimising the combined binary cross-entropy and Dice loss

$$\mathcal{L} = \mathcal{L}_{\text{BCE}} + \mathcal{L}_{\text{Dice}} \tag{3.1}$$

using the Adam optimiser with an initial learning rate of 10^{-4}. The learning rate was reduced by a factor of 0.5 if the validation loss did not decrease for five consecutive epochs. Training was run for 30 epochs with a batch size of 4, and the model weights with the lowest validation loss were saved as the final model.

3.2.5.4 Integration into the segmentation pipeline

After training, the U-Net replaces the Otsu binarization step in the **Frame** class. At inference time, each brightfield frame is padded to a multiple of 16 (the spatial reduction factor of the four pooling stages), passed through the network, and the output probability map is thresholded at 0.5 to obtain a binary mask. Connected components are then extracted from this mask and wrapped in **CellInstance** objects in exactly the same way as for the Otsu-based pipeline, ensuring that all downstream tracking and lineage analysis code is unchanged.

3.3 Establishing mother daughter relationship using positions and *Whi5*-activity

To establish a mother-daughter relationship, two factors are taken into account. Distance between cells and *Whi5*-activation.
Two factors were constructed to measure how likely it is that two cells are related. The two factors were multiplied to construct a $RelatedFactor$ estimating the plausibility that two cells are related. The cell with the highest related factor were assigned to be the mother cell.

$$RelatedFactor = distanceFactor \cdot whi5Factor \tag{3.2}$$

Both factors range from 0 to 1. Multiplying both factors will result in a $RelatedFactor$ that also range from 0 to 1. This $RelatedFactor$ will get a high value if both the $distanceFactor$ and the $whi5Factor$ have a high value. Another benefit with this representation is that one can scale the importance of these two parameters by changing the power of each factor. A higher factor will make a factor change faster making the factor more sensitive.
The distance factor was constructed with a modified sigma function. This can be seen as a smoothed step function.
The variable r represents the distance between the edges of the two cells at the moment of the potential daughter cells first detection.

$$distanceFactor = 1 - \frac{1}{1 - slopeFacor^{centerFactor - r}} \tag{3.3}$$

The $Centerfactor$ indicates where the factor reaches 1/2. This factor was set to equal 2.5 cell diameters. If the distance is larger then this, the cells are very unlikely to be related.
The $slopeFactor$ indicates the steepness of the slope at the center point. If this factor is very large, the function will approximate a step function. If this factor is small, the slope of the sigma function will be longer.
To quantify *Whi5* activity the fluorescent channel can be analyzed. When a cell divides and starts growing again, its *Whi5* activity increases. This *Whi5* activity can be quantified by observing the light detected at the fluorescent channel.
To quantify the amount of *Whi5* activity one can use different approaches. One way is to use the max intensity within the cell's area as measure for its *Whi5* activity. With x and y as pixel positions in the image, the $Whi5 - Activation$ can be written as:

$$Whi5 - Activation = \max_{x,y \in cell} I(x, y) \tag{3.4}$$

With this as the $Whi5 - Activation$ a cell is considered activated in a frame if the $Whi5 - Activation$ exceeds a certain threshold. The $whi5Factor$ is then written as the ratio of activated frames in a certain span after the potential daughter cell has been detected for the first time. This gives an idea of what cells are activated after the new cell is detected. However, sometimes the video does not make it clear whether a cell is activated or not. A minimum level of 0.10 was used for the $Whi5Factor$ so as not to completely disregard cells with no $Whi5 - Activation$ in the video.

$$Whi5Factor = \max(\frac{NumberActivatedFrames}{AnlysisSpan}, 0.1) \tag{3.5}$$

This gives a higher value for the cells activated in a large time span after the daughter cell is detected. If the cell is not activated in any of the frames after the daughter cell is detected the $Whi5Factor$ is set to the minimum value of 0.10.

The $RelatedFactor$ and its components are measured for every tracked cell, when a new cell is detected and the one with the highest $RelatedFactor$ is assigned as the mother cell.

3.4 Programming structure and dependencies

The program was implemented using Python and OpenCV as the image analysis framework. OpenCV is an open-source image analysis package with support for both Python and C++. OpenCV saves images in NumPy arrays and can be manipulated as such. It has functions for common operations such as thresholding, Gaussian blur, distance map, watershed algorithm and edge detection tools. There are also other available packages for image analysis. Sklearn is a popular library for machine learning that also contains methods for image analysis. This was used to create the previously mentioned random forest model.

3.4.1 Programming architecture and available classes

The architecture of the program is as follows. A main class which runs the program. Here methods for controls and loading data is called. Also a video object is created storing most of the functionality of the program (Figure 3.2)

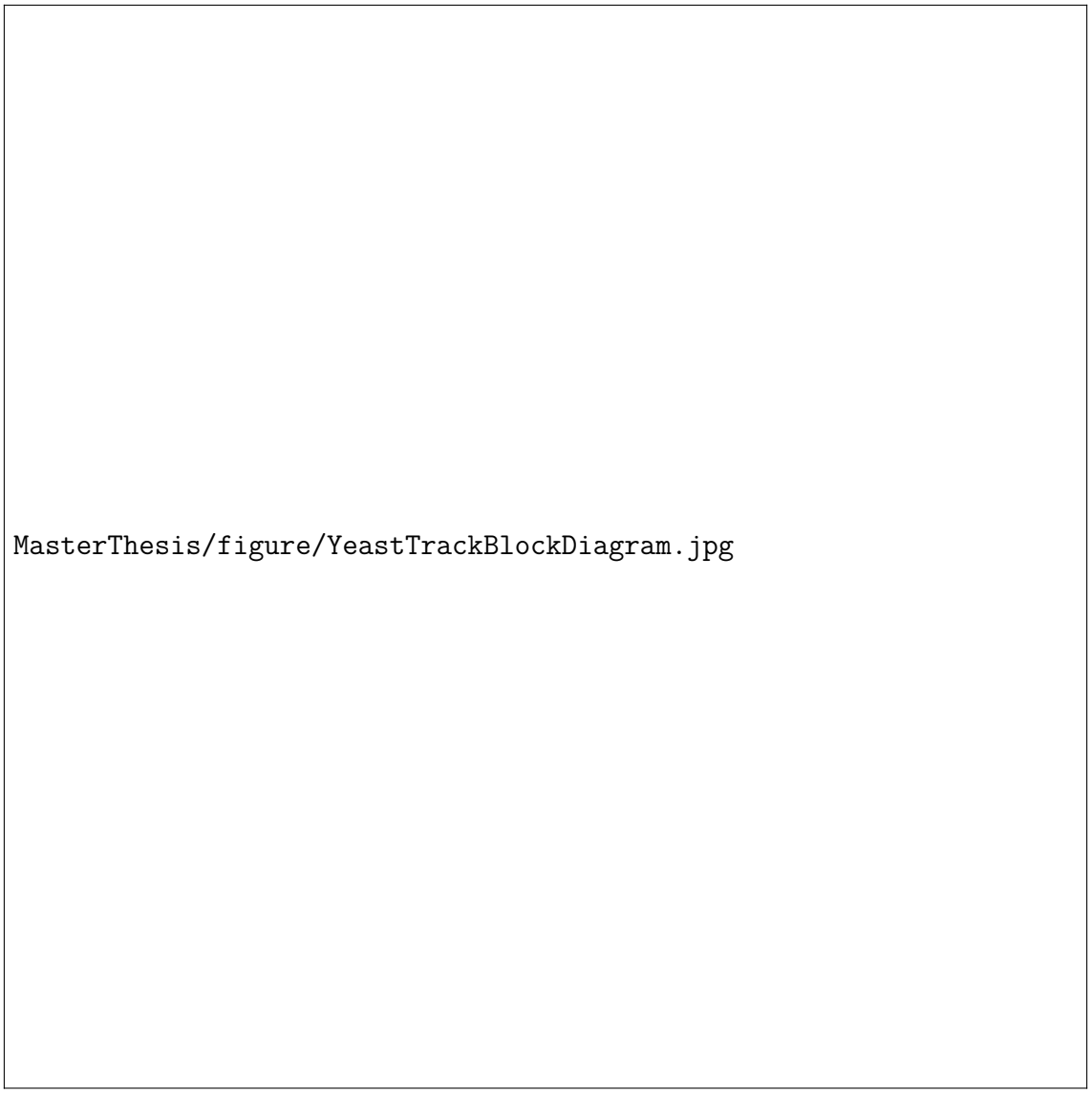

Figure 3.2: Block diagram describing the most important components of the program and how they relate to each other. Classes noted as **Class**, variables noted as variable, arrays noted as array[] and methods noted as method().

class Main
Calls method for loading data and setting up the visual environment.

class controls
This class contains the update method, which updates the displayed frame. But the class also contains the controls that allow the user to display different aspects of the process such as displaying segmentation areas, which cells are activated in *Whi5* and IDs at the centroid of each cell. There are also keys to run the methods contained in Analysis().

class LoadData

This class contains methods for converting *Leica Image File Format* to *.xml* files, as well as methods for loading *.xml* files and *.avi* files to the video class objects.

Class Frame
This class contains the images belonging to each frame. The images are the fluorescent image and the optical image. This class also contains methods for getting scaled images and images displaying the segmentation. Furthermore, this class contains the method for detecting cell instances. The **cellInstances** that are detected in the frame are saved in the **Frame** object.

Class CellInstance
This Class contains a momentary detection of a cell. It contains the points constructing the contour of the cell. It also contains methods for getting parameters such as size, position and *Whi5*-activity. These can be calculated using the contour points and the fluorescent image for the *Whi5*-activity.

Class TrackedCell
This Class contains an array of **cellInstances** belonging to the same cell. This array is put together by the **CentroidTracker** pairing the cell instances. It has methods that can generate growth curves and *Whi5* activity over time. It also contains a cellID as a label used for illustration and keeping track of mother-daughter relationships.

Class CentroidTracking
This class contains an indexed list with all tracked cells. It contains an update method that takes in an array of cellInstances and assigns them to the closest of the tracked cells according to the method explained in section 2.4. After the assignment it returns a list of all **TrackedCell** objects.

Class Video
This class contains an array of all frames, as well as an array containing all **TrackedCell** objects detected in the video. This class also contains the method for linage tracking. This method is run after the tracking has been completed and assigns each **TrackedCell** a mother cell.

4

Results

4.1 Image preprocessing by noise reduction and increasing contrast

In this section, the results from different preprocessing methods are presented. After evaluating available methods, a combination of median filter and histogram equalization was selected for preprocessing.

4.1.1 Noise removal by using filtering with Gaussian and median filter

The images were preprocessed by applying median filters with uniform weight and weighted by a sigma estimator. In both cases the noise was reduced but there was also a reduction in contrast. The filter weighted with sigma estimator resulted in less reduction of contrast, seen to the right in Figure 4.1. The uniform filter resulted in more reduction of contrast, seen in the middle of Figure 4.1.

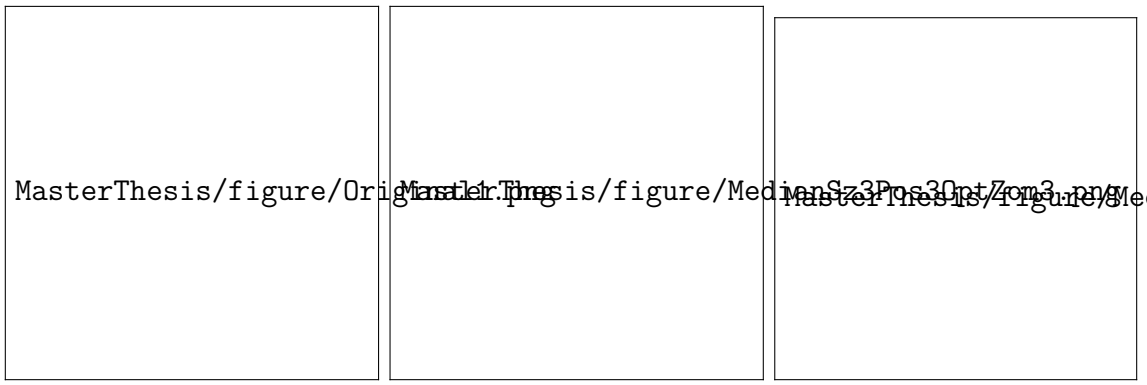

Figure 4.1: The original image for reference (left), a median filtering with size 3 (middle), median filtering with size 3 and weighted with sigma estimator (right).

The Gaussian filters were very effective in removing noise, but there was also a loss of information. Different sizes in the Gaussian kernel result in a different amount of information loss. With a kernel of size one the filter blurs the image but the edges are still distinguishable. With a kernel size of three it completely erases the edges and leaves just a faint mark where the cells were (Figure 4.2 right).

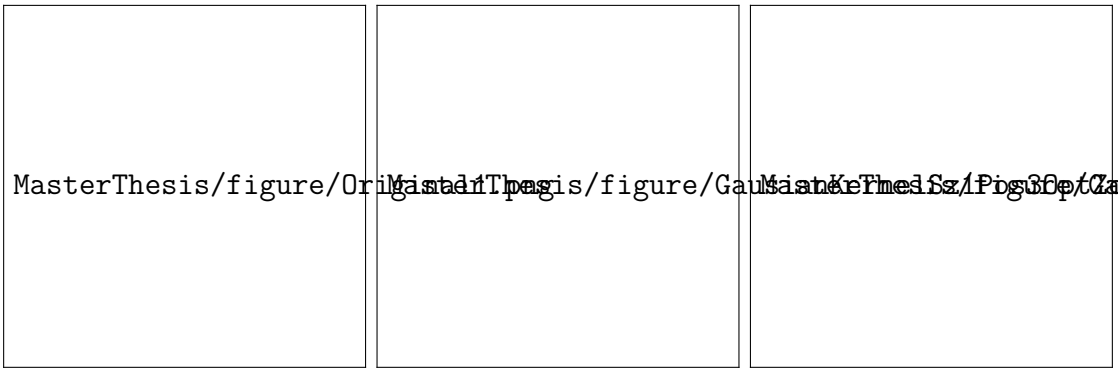

Figure 4.2: The original image for reference (left), Gaussian filter with kernel of size 1 (middle), Gaussian filter with kernel of size 3 (right).

4.1.2 Increasing contrast by using histogram equalization

After noise removal it is possible to make the objects more distinguishable by increasing the contrast in the image. The histogram equalization made the edges clearer and easier to distinguish in the optical channel (Figure 4.3 left).

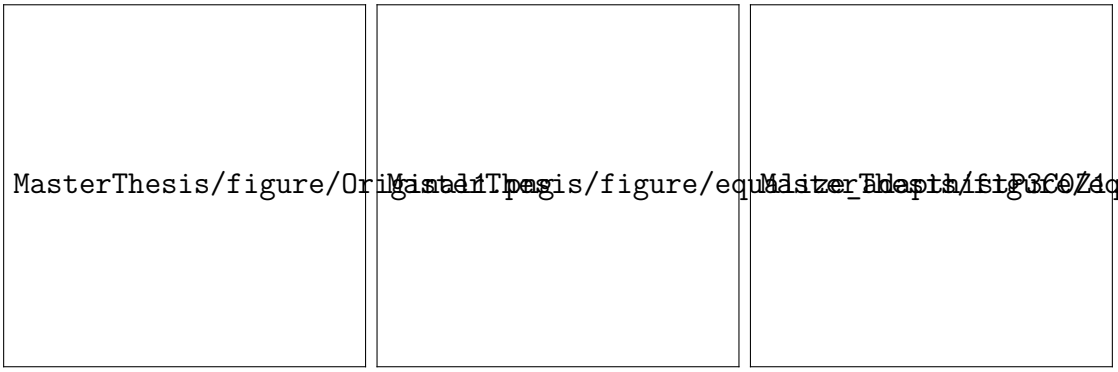

Figure 4.3: The original image for reference (left), equalizing histogram on the second focus level of the optical channel (middle), equalizing histogram on the forth focus level of the optical channel (right).

The histogram equalization did not have the same effect when applied to the fluorescent channel. A challenge in this data are the fluorescent pixels outside of the cells making the cells hard to distinguish from each other (Figure 4.4).

Figure 4.4: Original image for reference (left), equalizing histogram on second focus level of the fluorescent channel (right).

4.2 Results from segmentation of yeast cells

In this section results from different segmentation methods are presented. The different methods were segmentation with thresholding, Watershed algorithm, edge detection and random forest classifier.

4.2.1 Histogram analysis and segmentation of yeast cells using thresholding

The histogram of the images was analyzed using different threshold values. The red portion in figure 4.5 gave a clear segmentation of the edges of the cells. This portion had threshold values 0-45 applied to the first focus level. A similar result can be seen in figure 4.8 with threshold values 71-255 applied to the forth focus level. Both of these thresholds can be used to segment the cells by applying the convex hull. The two focus levels in the middle did not give an as clear representation of the edges and a much larger amount of the image is composed of intensities in the range 46-65 (Figure 4.6 and Figure 4.7).

Figure 4.5: Histogram from first focus level of the optical channel with marked thresholds for segmentation (left) and image illustrating segmentation from histogram with corresponding colors (right). The threshold values for the different colors were red: 0-45, green: 46-65, blue: 66-75 and yellow 67-255.

Figure 4.6: Histogram from second focus level of the optical channel with marked thresholds for segmentation (left) and image illustrating segmentation from histogram with corresponding colors (right). The threshold values for the different colors were red: 0-45, green: 46-65, blue: 66-75 and yellow 67-255.

Figure 4.7: Histogram from third focus level of the optical channel with marked thresholds for segmentation (left) and image illustrating segmentation from histogram with corresponding colors (right). The threshold values for the different colors were red: 0-30, green: 31-45, blue: 46-70 and yellow 71-255.

Figure 4.8: Histogram from forth focus level of the optical channel with marked thresholds for segmentation (left) and image illustrating segmentation from histogram with corresponding colors (right). The threshold values for the different colors were red: 0-20, green: 21-40, blue: 41-70 and yellow 71-255.

Figure 4.9: Histogram from fluorescent channel with marked thresholds for segmentation (left) and image illustrating segmentation from histogram with corresponding colors (right). The threshold values for the different colors were red: 0-6 and green: 7-255.

4.2.1.1 Results for Otsu-Binarization for segmentation of yeast cells

The Otzu-Binarization method was able to detect the edges of the cells. To later detect the entire cell the convex hull was applied. This method managed to segment the cells in the majority of cases (Figure 4.10 right). However, sometimes cells merged with each other causing errors in the detection. Therefore it was crucial to filter the detections according to size before submitting to the centroid tracking.

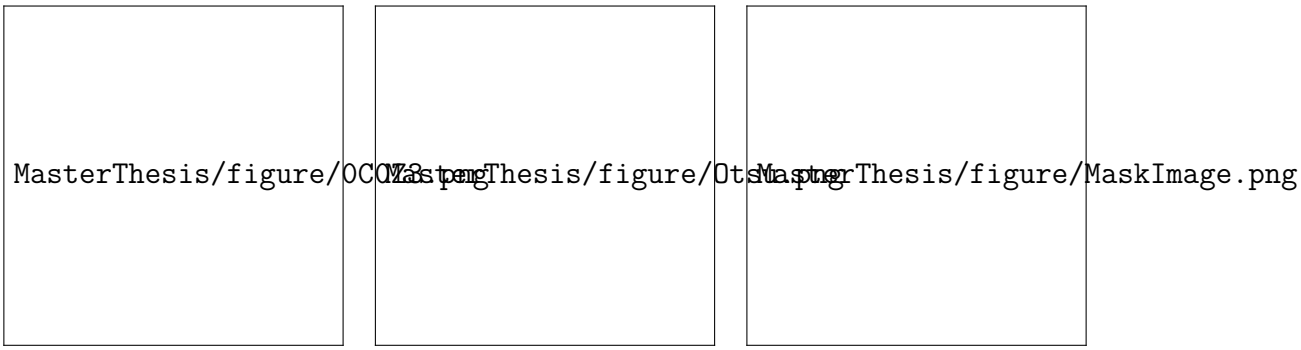

Figure 4.10: Figures illustrating process of segmenting with thresholding using Otsu-Binarization and convex hull. Original image for comparison (left). Result from Otzu-Binarization thresholding (middle). Image illustrating the final result when the convex hull is applied to the segmentation in the middle image (right).

4.2.2 Opening threshold, distance map and Watershed algorithm

In this section, the results from the different steps in the Watershed algorithm are presented. Different opening threshold images were tried. One opening threshold image was made with the fluorescent channel and others using the optical channel.

The distance map was able to separate every cell into local maxima in the distance map. These were then segmented using a threshold value. The different sizes of the cells result in difficulties in setting a threshold value that captures the centers of every cell while simultaneously not merging any of the cells. This is because the center of the cells has different values in the distance map depending on how large they are. The pixel value at the center is set depending on the distance to the background. In the end, a threshold value of 65% of the maximum value in the distance map was used. The process is illustrated in Figure 4.11. In this example, a noise detection has been registered to the right in the final result.

Figure 4.11: Figures illustrating the process of Watershed algorithm. Original image from optical channel (upper left) and image from fluorescent channel with histogram equalization for visibility (upper middle). Binary image illustrating the result from thresholding (upper right). This is the opening threshold. The result of the distance transform (lover left). This is a gray-scale image where the pixels in the center have higher values and the ones to the edges have a lover value. Figure illustrating a threshold applied to capture the center of the cells this threshold was set as 65% of the maximum value in the distance transform (lower middle). Figure illustrating the result of the Watershed algorithm starting from the segmented center point in the previous step and expanding outwards (lower right).

4.2.3 Edges detected using Laplacian Gaussian and Canny method

The edges became clearly distinguishable after the use of Laplacian Gaussian filtering. However, since the cells are packed so closely together it is hard to distinguish which edge belongs to which cell (Figure 4.12 middle).

The edges became clearer to distinguish once the Canny method was applied. However, the same problem with connecting edges where the cells are packed closely together is present (Figure 4.12 right).

In some cases, one can distinguish the cells as circles however when many cells are touching it is hard to distinguish the borders between each cell.

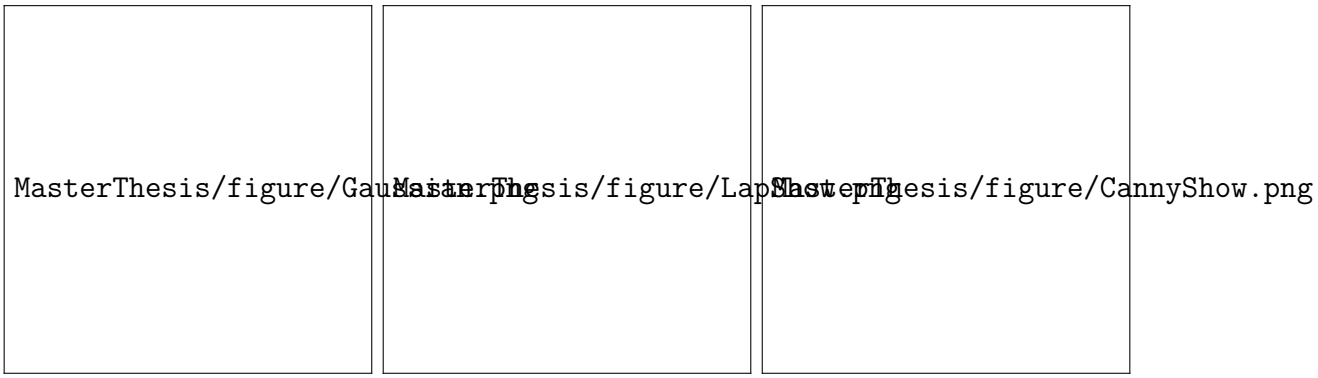

Figure 4.12: Example of Gaussian filter applied on optical channel (left). This blurs out potential noise in the image. Example of Laplacian applied to optical image with Gaussian blur (middle). Example of Canny method applied (right).

4.2.4 Random forest segmentation using annotated images

The random forest classifier classifies the majority of the pixels correctly (Figure 4.13 right). There are errors inside and outside the cells. But this approach also has a significant potential for improvement.

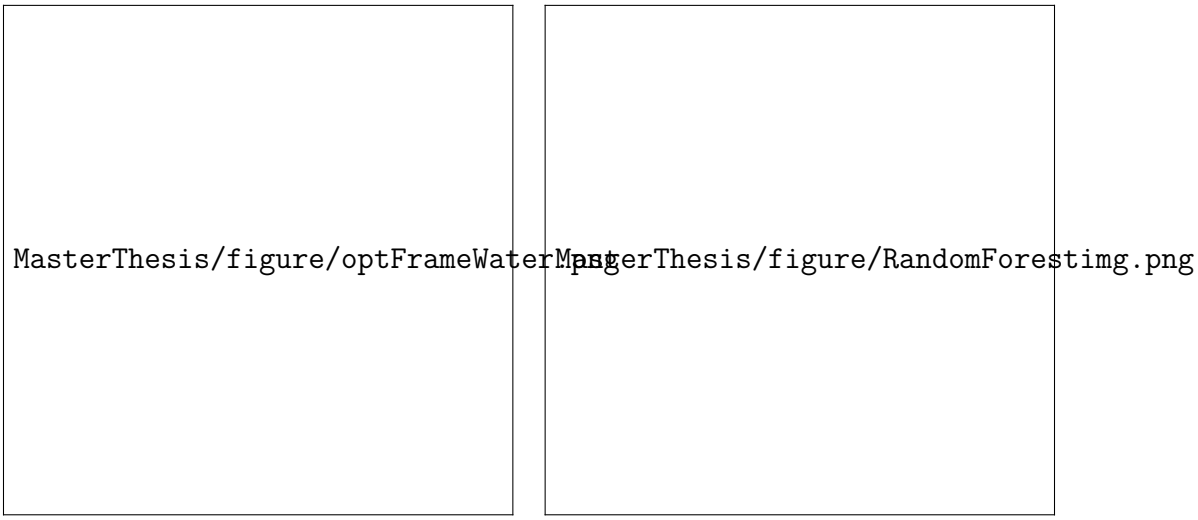

Figure 4.13: Example of Optical image from the test data set (left). Image classified with random forest classification trained on 7 manually annotated images (right).

4.2.5 U-Net segmentation trained on pseudo-labels

A U-Net was trained on all 157 brightfield frames of the *tileScan2* data set using binary masks generated by the Otsu binarization pipeline as pseudo-labels. Training was carried out for 30 epochs using the Adam optimiser with a combined binary cross-entropy and Dice loss.

Figure 4.14 shows the training and validation loss curves. Both losses decrease consistently from epoch 1 through epoch 30, with the best validation loss reaching 0.741 at the final epoch. The absence of a divergence between training and validation loss indicates that the model generalises well despite the small data set, partly due to the online data augmentation applied during training.

Figure 4.14: Training and validation loss (BCE + Dice) over 30 epochs. The model was trained on 157 brightfield frames with Otsu pseudo-labels. The best validation loss of 0.741 was achieved at epoch 30.

Figure 4.15 shows the brightfield input, the Otsu binarization result, and the U-Net prediction for the same frame. The U-Net produces a smoother mask with fewer isolated noise detections outside the cell colony compared to the Otsu result. Since the model was trained on Otsu pseudo-labels, the two outputs are similar in coarse structure, but the U-Net output is notably cleaner at cell boundaries and suppresses spurious background activations.

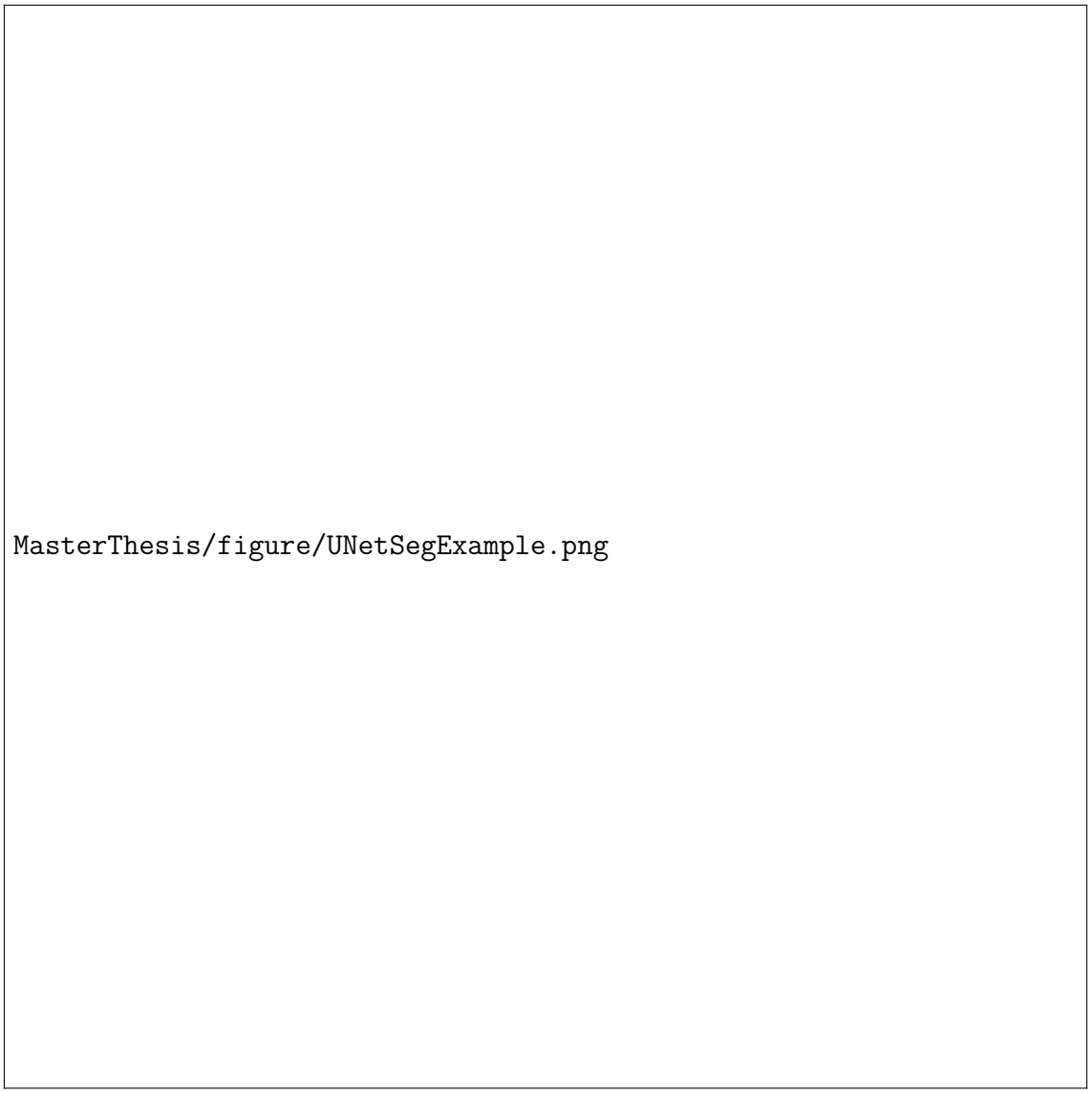

Figure 4.15: Comparison of segmentation methods on a brightfield frame. Left: original brightfield image. Middle: Otsu binarization with convex hull. Right: U-Net prediction after 30 epochs of training on Otsu pseudo-labels. The U-Net produces a cleaner mask with fewer spurious background activations.

4.2.6 Comparison between different segmentation methods

All segmentation algorithms were compared with each other and the ones giving the most promising results were implemented in the program (Figure 4.16). The ones available in the program are segmentation with Otsu-Binarization, Watershed algorithm, random forest, and U-Net.
The segmenting algorithm giving the best result is the Otsu-Binarization (Figure 4.16 upper left panel) but it lacks possibilities for further development. The random forest classifier is the algorithm with the most potential for further improvements among the classical methods (Figure 4.16 lower right panel). The U-Net approach extends the pipeline with a learnable segmentation stage that can be retrained on

new data or improved with manual annotations.

Below in figure 4.16 is an example where all segmentation methods are applied to the same image. The segmentation using Otsu-Binarization works for all cells in this image. The Watershed algorithm classifies some of the cells correctly but some have merged and some have avoided detection altogether (Figure 4.16 upper right panel). The Canny method makes the edges clearly distinguishable but it is hard to separate them in this image (Figure 4.16 lower left panel). Lastly, the random forest classifier finds the cells but the center of the cells still contains a large number of errors (Figure 4.16 lower right panel).

Figure 4.16: Figure illustrating all the investigated segmentation methods for comparison. Otsu-Binarization and convex hull (upper left). Image classified with Watershed algorithm (upper right). Image with Canny method applied (lower left). Image classified with random forest classifier (lower right).

4.3 Tracking trace produced by centroid tracking

It was possible to track the cells through the video, results illustrated in figure 4.17. However, when the segmentation failed the tracking stopped at the latest registered positions until the segmentation found the object again. Since the cells had very limited motion a simple centroid tracking algorithm worked well for tracking the cells. In some cases, there were problems with cells not being detected. This is a problem with the segmentation rather than with the tracking, however it does affect the tracking performance. Other problems caused by errors in the segmentation are cells switching positions. That is one cell from one trace registers a detection from another cell nearby, thus switching positions and sizes.

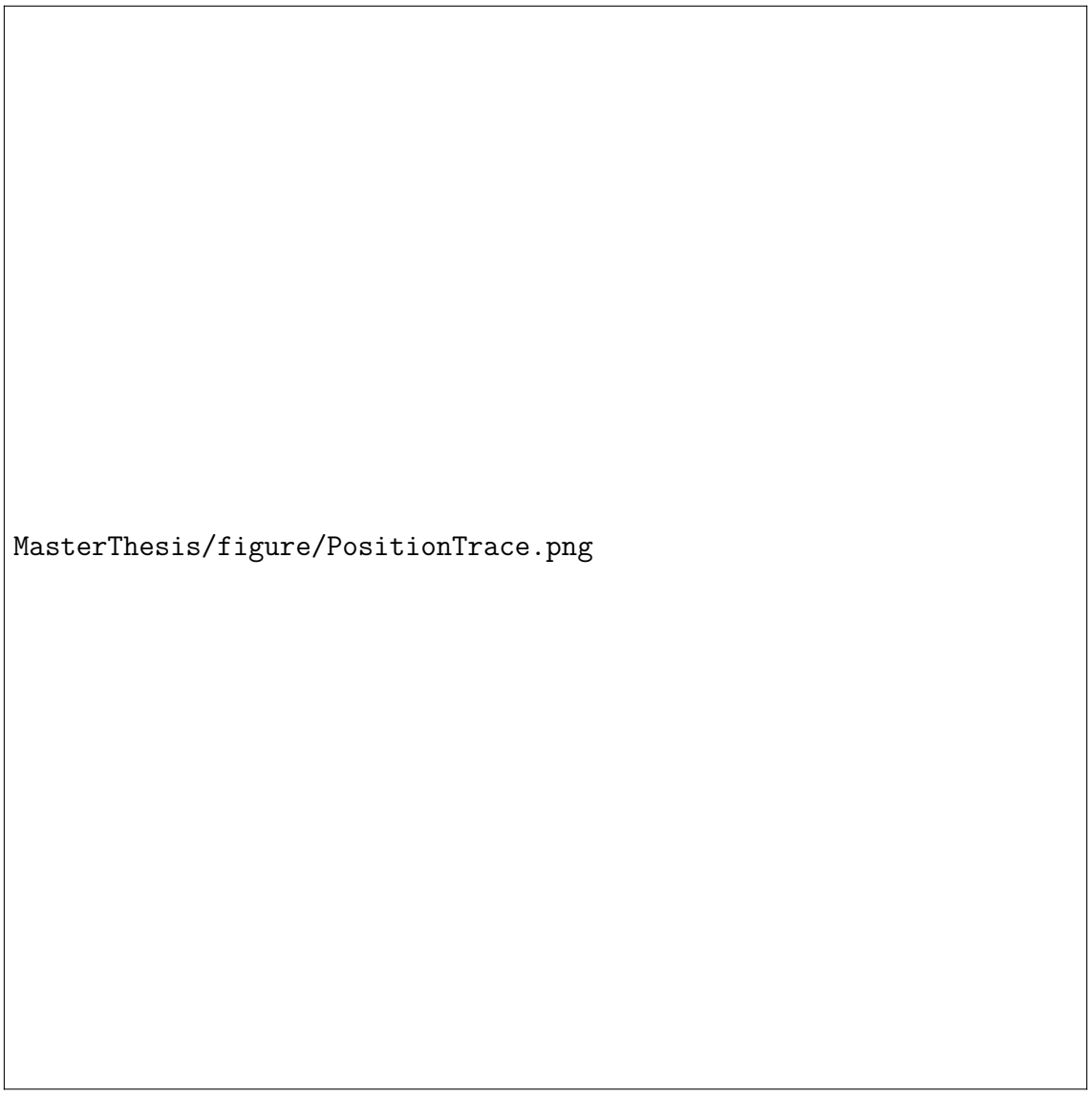

Figure 4.17: Positions for cells throughout the video. Each cell trace is plotted with an individual color.

4.4 Lineage Tree and *Whi5* activity

Whi5 activity was extracted for selected cells from the fluorescent channel. When analyzing the *Whi5* activity, it has been observed that the spiking of *Whi5* was not always synchronized between mother and daughter cells (Figure 4.18 left).

Figure 4.18: Two examples of extracted *Whi5* activity for a linage. To the left, cell with ID 0 is the mother of both cell ID 6 and ID 13. The spikes are clearly related. To the right, cell with ID 7 is the mother of cell with ID 11. The spike in *Whi5* Activity is not related.

The lineage tree constructed by the program was plotted as a tree graph. One graph was plotted using the information of *Whi5* (Figure 4.19 right and Figure 4.20) and one without any *Whi5* information (Figure 4.19 left and Figure 4.21). In the latter case the $Whi5Factor$ is set to a constant for all cells. In the lineage trees with the *Whi5* information the relationships tended to aggregate towards the cells who were spiking in *Whi5*. Without *Whi5* information the cells were assigned to the nearest tracked cell.

Figure 4.19: Ilustation of two small linage trees. Edges are labeled with *RelatedFactor*. Linage tree with no *Whi5* information is used (left). Linage tree with *Whi5* information is used (right).

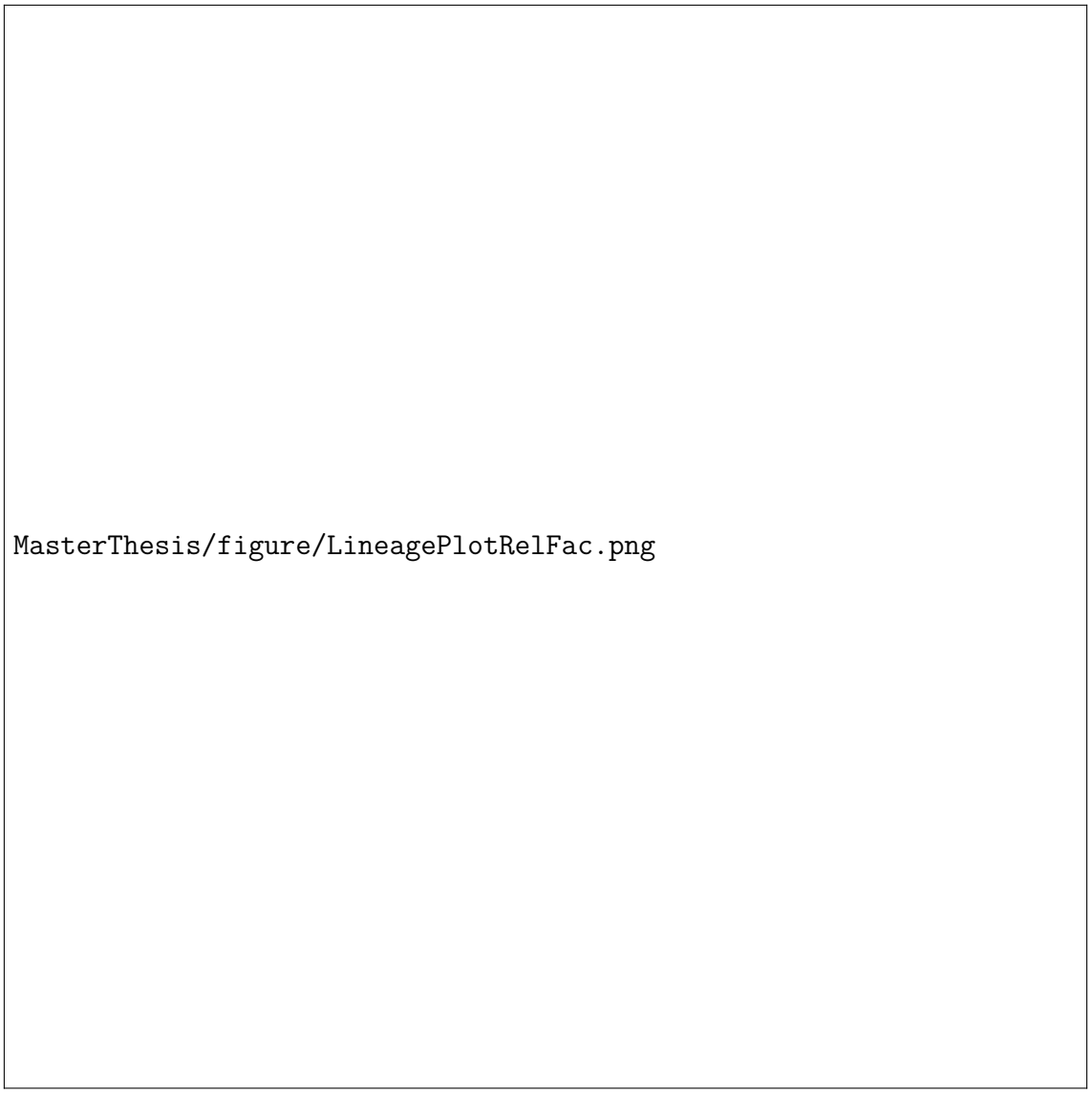

Figure 4.20: Large linage tree plotted with no *Whi5* information used. Edges are labeled with the $RelatedFactor$.

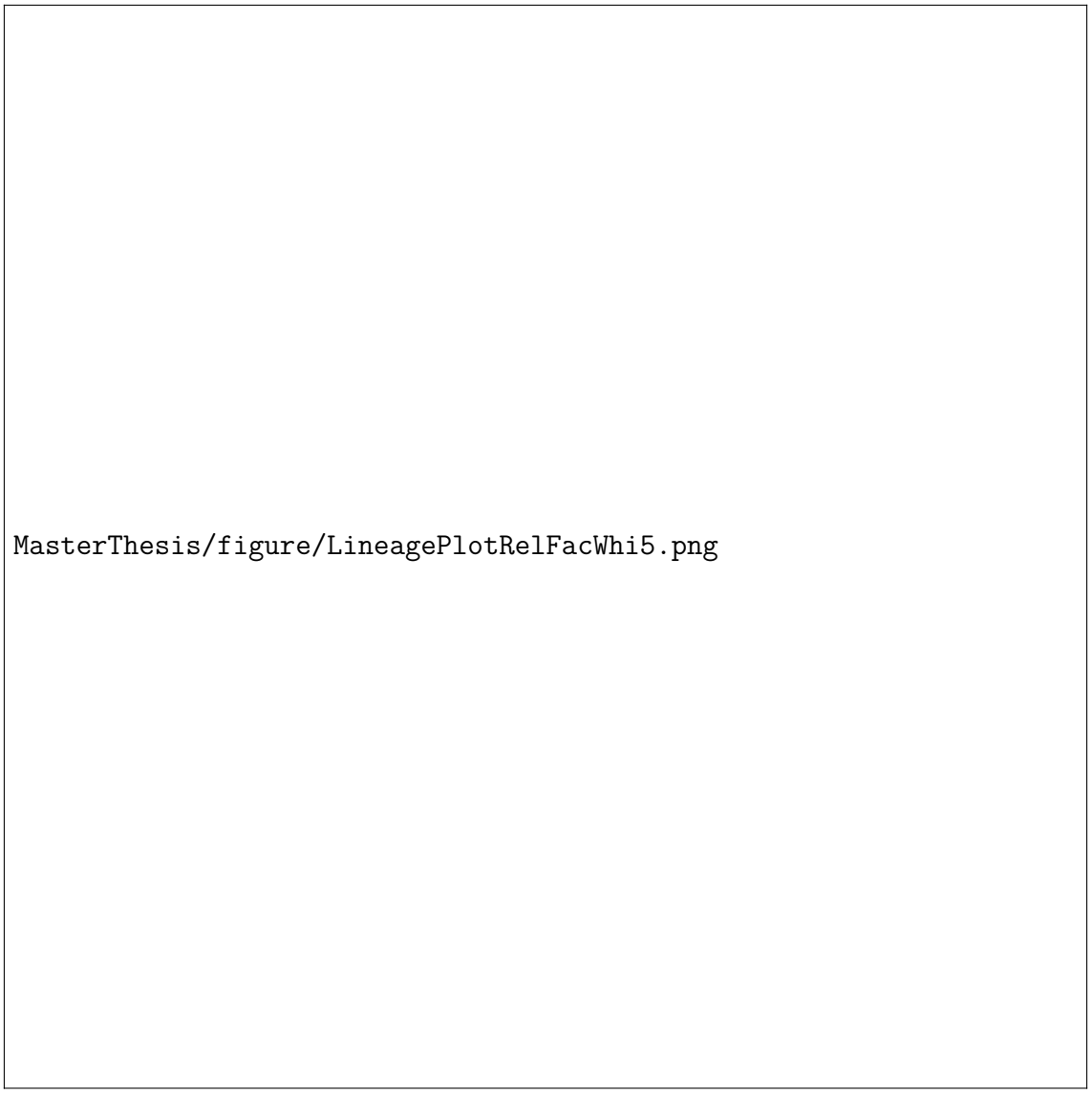

Figure 4.21: Large linage tree plotted with *Whi5* information used. Edges are labeled with the $RelatedFactor$.

4.5 Growth curves using segmented area and daughter cells

The size of the cells was extracted and plotted in growth curves (Figure 4.22). The cell size is calculated as the size of the mother cell added to the size of the bud, which is the daughter cell. This is the case until the bud separates from the mother cell indicated by the *Whi5* activity.

Figure 4.22: To the left is example of extracted data. This is growth curves for a linage where Cell with ID 0 is the mother of both cell ID 6 and ID 13. To the right is cell size where bud size have been added to mother cell. The dotted line indicate cell division. This is the point were the bud size stops being added to the mothers size.

5
Conclusion

In this work a pipeline that is able to extract cell lineages from microscopy images has been constructed. To achieve this, several different segmentation methods were investigated. A simple thresholding approach managed to segment cells, but other methods, like the random forest classifier, need further improving. As an extension of the original work, a deep learning segmentation method based on the U-Net architecture was implemented and trained, providing a learnable alternative to classical thresholding. To trace the cells through the video, a centroid tracking algorithm was implemented enabling a cell tracking upon successful segmentation. To measure how likely a new cell is to be related to each one of the other tracked cells, we defined a *RelatedFactor*. This factor gave a probable prediction. However, the predictions can only be as good as the tracking and segmentation.

5.1 Evaluating segmentation algorithms and exploring possible development

A Otzu threshold worked for segmenting the edges of the cells since these were clearly distinguishable from the background. This makes it easy to detect the whole cell by filling the center and then applying the convex hull. However, there were many invalid detections and the threshold is sensitive to other objects than cells. This can be handled by having clean video data without distracting objects. One should also apply filtering of detections before registering them for tracking.
There are opportunities for improvement in this method. Since this threshold method only looks at one channel and one focus level, one could improve it substantially by also taking the other focus levels and the fluorescent channel into account. These other focus levels and the fluorescent channel could be analyzed with individual threshold values, creating several classified images. To combine the different binary images, one could use methods such as consensus classification.

The Watershed algorithm did not manage to classify the majority of cells correctly in the current implementation. The first challenge is finding a good opening threshold. The investigated thresholds did not have enough separation between the cells, or there was too much noise to give a clear distance map.
Another challenging aspect is the different sizes of the cells. The different sizes gave different values for the center of the cells. This made it challenging to find a good value for the thresholding of the centers. If the value is too high, it does not manage to detect the smaller cells. If the value is too low, it merges the larger closely packed

cells.
One way to find the centers without thresholding could be by finding maximum points in the image using for example gradient decent. The maximum points would correspond to the centers of the objects.

The Laplacian Gaussian filter was implemented successfully. However, since the cells are closely packed their edges were merged, making them hard to distinguish from each other. After the Canny method was applied the edges became significantly clearer, but there were still problems with merging edges. To develop this method further, one could vary the limits of what is considered an edge. To continue developing the method, one could also try to segment the cells using a convex hull or identifying ellipses in the image.

The random forest classifier gave promising results. It would be interesting to incorporate methods for testing the accuracy of the model, guiding further improvements. The model gave fewer errors when more data were added. One would want to see how long this trend continues. Another improvement that could be made is by combining several different classifications. One could for example add the result from classification with thresholding as a parameter to the random forest algorithm. This gives the classifier more information to improve its performance. Other parameters could be added like the distance map and more filters could be applied to the original images. One could then use the method for testing accuracy to see which parameters are contributing significantly to the result. The redundant parameters can be removed to save memory space and running time.
There are also improvements that can be done post-classification, for example removing small noise and filling small gaps. This can be done by using growth and erode functions: first shrinking the segmentation to remove single pixels using erode, and then expanding the segmentation using grow to fill any gaps.

The U-Net segmentation method was implemented and trained on Otsu pseudo-labels generated automatically from the brightfield video frames. The model converged consistently over 30 epochs, reaching a best validation loss of 0.741, and produced cleaner segmentation masks than the Otsu baseline it was trained on. Replacing Otsu with U-Net in the pipeline is straightforward since the two methods share the same interface. The primary avenue for further improving the U-Net is to retrain it on a small set of manually annotated frames — even 20–30 carefully labelled images would be expected to yield a substantial improvement over pseudo-label training.
To investigate the segmentation further, other segmentation methods based on convolution could be explored. All the investigated classical segmentation methods classify each pixel independently. A segmentation algorithm that uses convolution can capture the circular shape of the cells and learn spatial context, helping make more accurate classifications. Another method that requires less labelled data is to use some convolutional layers that are then fed to either a random forest classifier or a support vector machine [2].

5.2 Exploring results and obstacles of centroid tracking

The tracking algorithm worked well in the majority of cases. However, in some cases cells could not be detected and in some cases cells switch places with other nearby tracked cells. Therefore, better segmentation would entail better tracking. But there are improvements that could be made to the tracking algorithm, for example by applying further conditions on connecting detections to tracked cells. In particular, one could put conditions on the distance and size differences between the tracked cell and the detection. If the detection has a position and size that meet the conditions it is accepted, and if it does not meet the conditions it will be discarded. Other tracking algorithms could also be investigated for comparison.

5.3 Extracted data and Limitations in the Data

The sizes, positions and *Whi5*-activity were accurate in the majority of the tested cases. However, the connection between mother and daughter cell was not reliable.

5.3.1 Lineage tree and related factor

The linage was extracted but there were often many possible candidates for the mother cell. The approach of using a factor to measure the suitability of assigning mother to daughter cell proved successful when the basic information like position and *Whi5*-activity were correct.

The approach of using the portion of frames were the *Whi5*-activity spikes also proved to be the best representation of *Whi5*-activity of the ones that were investigated.

One possible improvement would be to use the information of the time to the latest division. If a cell has been dividing recently, it is unlikely that it has divided again when the new cell is detected. Including this in the factor would further exclude possible mother cells, making the linage tree more accurate.

5.3.2 Growth curves

The growth curves did not exhibit any clear relationship except increasing size. To make it easier to analyze, one could make a clearer definition of what constitutes a cell and exactly when a cell divides from its mother. One should also note that these are two-dimensional pictures of three-dimensional objects, which means that even in the best case scenario one gets only a rough approximation of the size. To make it more accurate, one could try to approximate the three-dimensional size with an ellipsoid. This could give a clearer insight into the nature of the growth curves.

5.3.3 Limitations in the data

There were limitations to the data. Some are due to the quality of the fluorescent channel and some are due to the resolution.
It is not always clear which cell is the mother cell because the *Whi5* spike is not always noticeable in the video. This makes it less clear-cut which cell is the mother cell if there are many cells with equal distance to the daughter cell.
Another problematic situation is if many nearby cells spike at the same time. That makes it unclear which one of them is the mother cell. In both of these cases, more information such as time from the latest division would enable a more accurate assignment.

A clearer fluorescent channel would increase the accuracy of the measured *Whi5* activity. An increased resolution could potentially make the fluorescent channel more clear. This would also make the separation between the cells larger, measured as the number of pixels. A larger separation between the cells would make it more unlikely for segmentation areas from different cells to merge.

5.4 Programming architecture and further development

The structure of the program worked well for the tasks. However, there were some memory issues when trying to load all focus levels for the frames.
This can be because of inefficient memory allocation or the fact that Python is not suited well for scalable projects. If the program would be expanded further, one might want to consider translating to another programming languages such as C++.
To address the memory issue, it would also be reasonable to develop the frame class and loading methods, which is the class for holding and loading the images. To allocate the images in an efficient way in this class, it is crucial for the program to run fast.
One should also note that all parameters are hardcoded in the program. This could be problematic if a video of another kind would be analyzed. For example, if a video with higher resolution would be used, the cells would appear larger and therefore be filtered from detection. Consequently, a better approach would be to let the user set these values or set them by auto-detection.

This program is a start for a useful alternative analysis tool to the ones that are already available. This would open up possibilities for implementing other tools available in the Python environment, by simplifying the development step and also giving possibilities to implement further analysis tools.

Bibliography

[1] In: ().

[2] Y. W. Y. C. Aili Wang. "Hyperspectral image classification based on convolutional neural network and random forest". In: (2019). https://www.tandfonline.com/doi/citedby/10.1080/2150704X.2019.1649736?scroll=top&needAccess=true.

[3] G. Aubert, R. Deriche, and P. Kornprobst. "Computing optical flow via variational techniques". In: *SIAM Journal on Applied Mathematics* 60.1 (1999), pp. 156–182.

[4] C. B. Bernhard Preim. *Watershed Segmentation*. https://www.sciencedirect.com/topics/computer-science/watershed-segmentation. 2014.

[5] G. Bradski and A. Kaehler. "OpenCV". In: *Dr. Dobb's journal of software tools* 3 (2000). https://opencv.org/.

[6] J. Canny. "A Computational Approach to Edge Detection". In: (1986). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.420.3300&rep=rep1&type=pdf.

[7] W. Chen et al. "A Review of Biological Image Analysis". In: *Current Bioinformatics* 13 (July 2018). https://www.researchgate.net/publication/326297438_A_Review_of_Biological_Image_Analysis, pp. 337–343. DOI: 10.2174/1574893612666170718153316.

[8] D. Comaniciu and P. Meer. "Mean shift analysis and applications". In: 2 (1999), pp. 1197–1203.

[9] G. Deng and L. Cahill. "An adaptive Gaussian filter for noise reduction and edge detection". In: (1993), pp. 1615–1619.

[10] D. L. Donoho and I. M. Johnstone. "Ideal spatial adaptation by wavelet shrinkage". In: *Biometrika* 81.3 (1994). https://doi.org/10.1093/biomet/81.3.425, pp. 425–455. DOI: 10.1093/biomet/81.3.425.

[11] O. M. Environmen. *Bio-Formats*. https://www.openmicroscopy.org/bio-formats/. 2020.

[12] "Fiji". In: (2020).

[13] "Github for MachineLearning For Segmentation". In: (2020). https://github.com/Klas96/MachineLearningForSegmentation.

[14] "ImageJ". In: (2018).

[15] T. Itoh, H. Sueda, and Y. Watanabe. “Motion compensation for ISAR via centroid tracking”. In: *IEEE Transactions on Aerospace and Electronic Systems* 32.3 (1996), pp. 1191–1197.

[16] J. S. Jean-Yves Tinevez Nick Perry. *TrackMate*. `https://imagej.net/TrackMate`. 2020.

[17] T. P. Jean-Yves Tinevez. “MaMuT”. In: (2020). `https://imagej.net/MaMuT`.

[18] H. Kong, H. Cinar Akakin, and S. Sarma. “A Generalized Laplacian of Gaussian Filter for Blob Detection and Its Applications”. In: *Cybernetics, IEEE Transactions on* 43 (Jan. 2013), pp. 1719–1733. DOI: `10.1109/TSMCB.2012.2228639`.

[19] F. Mendoza and R. Lu. “Basics of Image Analysis”. In: *Food Engineering Series* (Jan. 2015), pp. 9–56. DOI: `10.1007/978-1-4939-2836-1_2`.

[20] A. Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. `https://pytorch.org`. Curran Associates, Inc., 2019, pp. 8024–8035.

[21] O. Ronneberger, P. Fischer, and T. Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention* (2015). `https://arxiv.org/pdf/1505.04597.pdf`.

[22] B. Schnitzer, J. Borgqvist, and M. Cvijovic. “The Synergy of Damage Repair and Retention Promotes Rejuvenation and Prolongs Healthy Lifespans in Cell Lineages”. In: *bioRxiv* (2020). DOI: `10.1101/2020.03.24.005116`. eprint: `https://www.biorxiv.org/content/early/2020/03/25/2020.03.24.005116.full.pdf`. URL: `https://www.biorxiv.org/content/early/2020/03/25/2020.03.24.005116`.