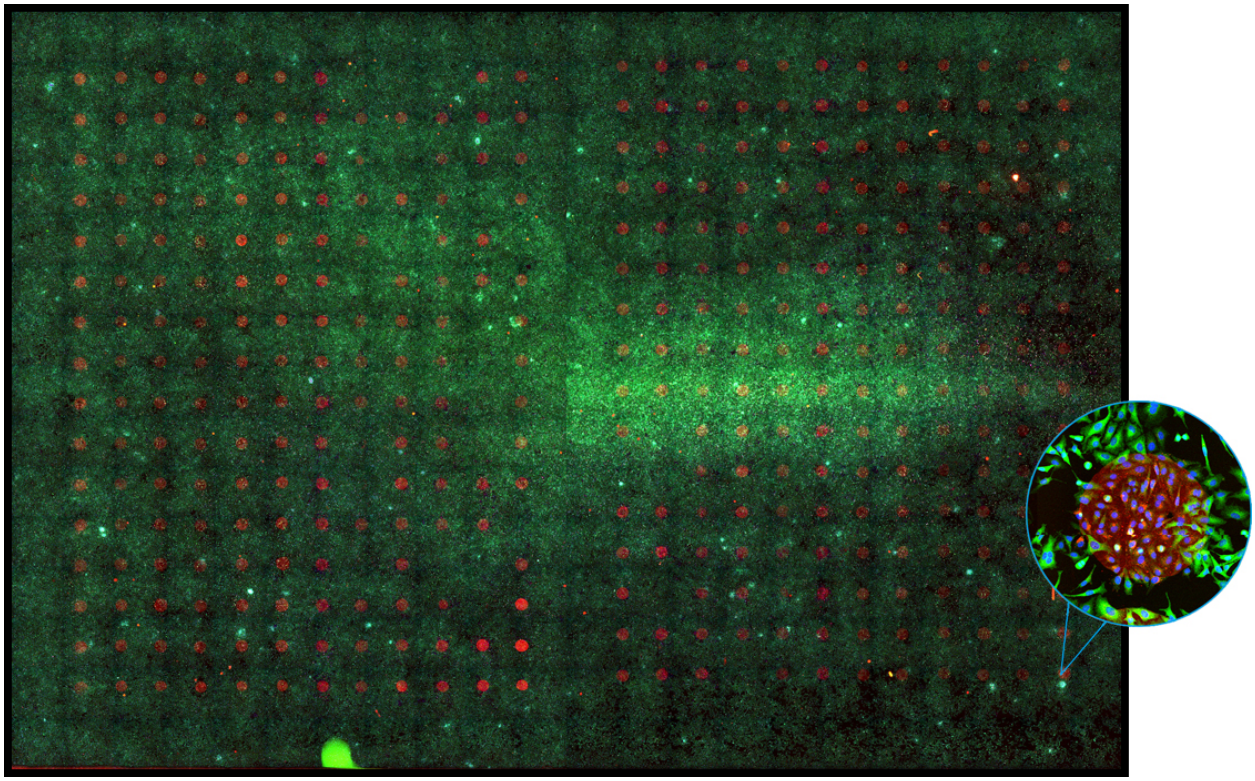
CHALMERS
UNIVERSITY OF TECHNOLOGY



Automatic Spot Detection in Large-Scale Fluorescence Microscopy Image Datasets

Master's thesis in Systems, Control and Mechatronics

MARCUS BJÖRKLUND

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Master's thesis EX062/2017

Automatic Spot Detection in Large-Scale Fluorescence Microscopy Image Datasets

MARCUS BJÖRKLUND

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals and Systems
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2017

Automatic Spot Detection in Large-Scale Fluorescence Microscopy Image Datasets
MARCUS BJÖRKLUND

Supervisors: Viktor Bengtsson, Persomics
Fredrik Kahl, Department of Signals and Systems
Examiner: Fredrik Kahl, Department of Signals and Systems

Master's Thesis EX062/2017
Department of Signals and Systems
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A typical downsampled channel-optimized montage of an array image screened by Persomics, with one of the experiment spots enlarged.

Typeset in LaTeX
Printed by [Name of printing company]
Gothenburg, Sweden 2017

Automatic Spot Detection in Large-Scale Fluorescence Microscopy Image Datasets
MARCUS BJÖRKLUND
Department of Signals and Systems
Chalmers University of Technology

Abstract

In this thesis, we consider the problem of detecting spots that encapsulates biological experiments printed on array plates. Our primary goal is to automatically detect and annotate all individual experiment spots in fluorescence microscopy images for further quantitative analysis. The proposed approach can be summarized as follows. Initially, image data is decimated, enhanced and montaged into a single composite image. Then, a k-NN classifier is used to segment the spots and sub-arrays are detected with a k-means clustering approach. Each sub-array is analyzed individually and a new gridding algorithm yields the automatic segmentation of the array image into spot quadrants. In contrast to other methods, the gridding is based on intensity profiles. Finally, the spots are detected within each quadrant by thresholding and by computing centroids of connecting components. An extensive evaluation on several data sets with ground-truth has been carried out. The results indicate that our method efficiently detects the spots in array images and outperforms the currently used method, which relies on pre-saved annotation coordinate data.

Keywords: fluorescence microscopy, spot detection, large-scale images, image segmentation, image classification, array gridding

Acknowledgements

I would like to thank Martin Svensson at Persomics for giving me the possibility of working on this thesis and for the many interesting discussions about the implementation of the spot detection framework. I am especially grateful for the help and guidance provided by my supervisor at Persomics, Viktor Bengtsson. I highly appreciate his help, inspiring discussions and feedback during my time at the company. I would especially like to thank my SO for incessantly motivating and supporting me throughout the many months of work. Special thanks also go to my closest friend, Jakob Lundegard, for proofreading the thesis. Finally, I would like to thank Fredrik Kahl, my supervisor and examiner at Chalmers University of Technology, for the feedback and guidance.

Marcus Björklund, Gothenburg, May 2017

Contents

Glossary

Array image is the whole set of component images from an array plate screening.
Array plate is a glass plate with printed gene silencing experiments.

Component image is a non processed tile image acquired from a fluorescence microscope at a certain wavelength, row and column position.

Montage image is the whole set component images in an array image, montaged into a single composite image.

Print map is an exact layout definition of the experiment spots in a certain print number.
Print number is the name of a certain print map and each array plate has one.
Print type is how an array plate is printed. Single-sided types are printed in one contact whilst double-sided types are printed in two contacts, where the seconds contact is rotated 180° before printing.

Spot image is a multichannel image containing a single experiment spot, generated by the detection framework.

List of Figures

List of Tables

1

Introduction

In this thesis, the problem of detecting spots that encapsulates biological experiments printed on *array plates* is considered. The aim of the project is to automatically detect and annotate all individual experiment spots in fluorescence microscopy images so they can be used for further quantitative analysis.

1.1 Background

An *array plate* is a technology developed by Persomics [1] that allows scientists to perform thousands of parallel and unique cell biological experiments in an easy to use consumable instead of using expensive robotic labs [2]. The gene silencing experiments can printed in 350 μm spots [1] on a glass plate as the example provided in Figure 1.1 and up to 3200 experiments can be printed on a single array. Gene silencing is an important research tool but has up to this point been reserved for large and well-funded laboratories [2]. This research tool can be democratized and be brought up to the wider research community when an automatic detection of the experiment spots is attained.

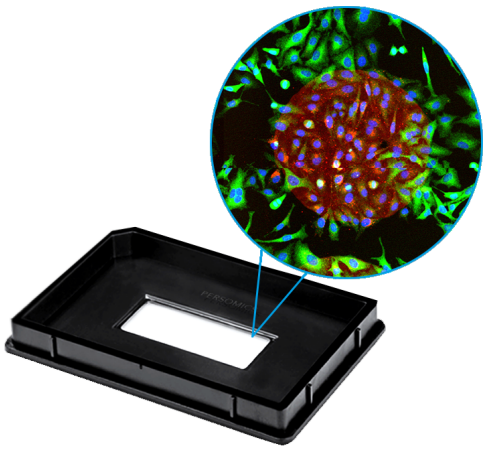


Figure 1.1: An *array plate* with one of the experiment spots enlarged and enhanced. From Persomics [1], used with permission.

1.1.1 Array Plate

Each *array plate* has an attribute *print number*, which is the name of a certain *print map*. The *print map* of an *print number* is the exact layout definition of the experiment spots. It holds information about the radius r of each spot, the horizontal (d_h) and vertical (d_v) spacing between spots and the number of spot rows and spot columns. An example *print map* is illustrated in Figure 1.2.

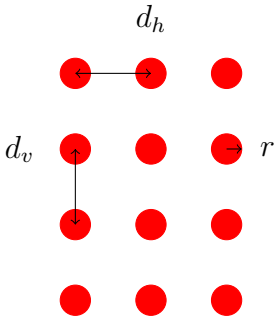
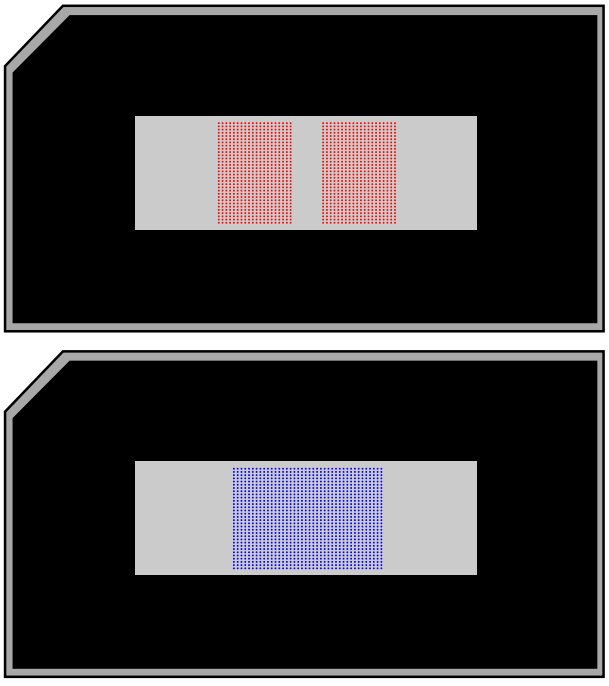
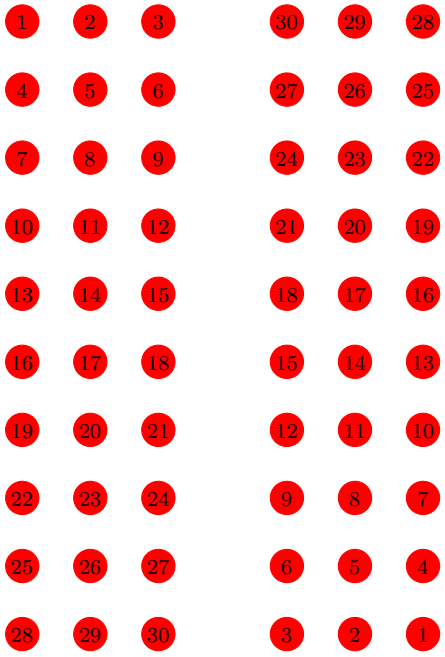


Figure 1.2: An example of a *print map* with three columns and four rows. Each *array plate* is defined by a *print map* with different horizontal (d_h) and vertical (d_v) spacing, where each printed spot has a spot radius r.

The *print map* also specifies the *print type* of the *array plate*. There are currently two *print types*, namely single and double-sided *array plates* as the example in Figure 1.3a. Single-sided types are printed in one single contact, whilst double-sided types are printed in two contacts, where the second contact is rotated 180° before printing, as the example in Figure 1.3b.



(a) There are currently two *print types*, double and single-sided. The top array is a double-sided *array plate* and the bottom array is a single sided *array plate*.



(b) An example of a double-sided array plate. The left sub-array has the same content as the right one and the right sub-array is always rotated 180° compared to the left one.

All Persomics' *print maps* have the same spot radius, horizontal and vertical spacing with the exception of two prints, p74 and p86 whose vertical and horizontal spacing

are doubled. The three printing parameters are specified in Table 1.1 for all *print numbers*.

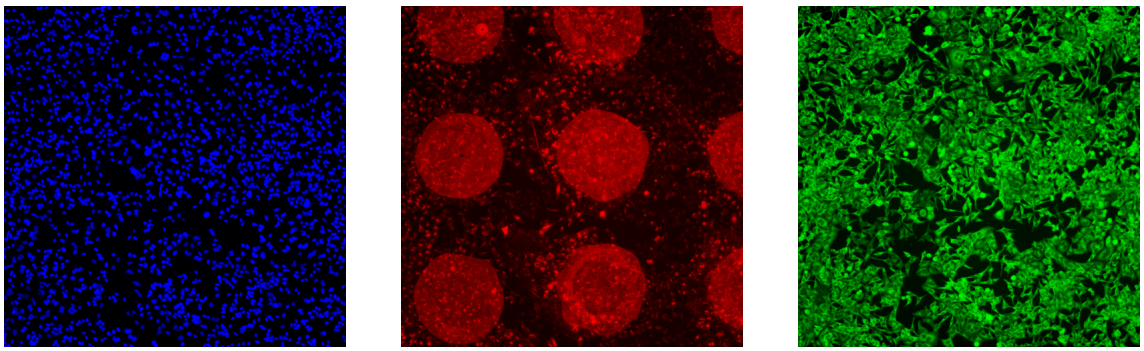
Print number	r	d_h	d_v
p74, p86	350 μm	1100 μm	1000 μm
p67->p73, p75->p85	350 μm	550 μm	500 μm

Table 1.1: The spot radius and the vertical and horizontal spacings between spots in all *print maps*. Two prints p74 and p86 have double spacings.

Persomics supports *array plates* with rows in range $\in$ [16, 32] and columns in range $\in$ [2, 100]. The currently smallest printed array is 32 rows x 2 columns and the largest one is 32 rows x 38 columns.

1.1.2 Screening Procedure

Each printed spot in an *array plate* is labeled with a red dye. They also encapsulate an RNA and other reagents that are needed for silencing a gene inside the cells that are added to the experiment. The initial step is to add cells on top of the spots and incubate between 48 to 120 hours [2]. The cells are then stained with a green fluorescent antibodies dye and a blue hoechst dye. This colour labeling is used so each cellular component easily is identified. The whole array of spots is then screened by a *fluorescent microscope* at each dyes' emission wavelength. Since a high resolution image with a large field of view is desired, the whole array can not be imagined in one step due to hardware limitations. The region of interest is subdivided into multiple smaller sub-regions which are imaged individually, generating *component images*. Figure 1.3 shows an example of three different *component images* screened at the same position but at three different wavelengths. A more detailed description over the screening procedure can be found in the screening protocol in [3].



(c) *Component image* screened at the wavelength where the blue hoechst dye emits light.

(d) *Component image* screened at the wavelength where the red dye emits light.

(e) *Component image* screened at the wavelength where the green antibodies dye emits light.

Figure 1.3: Three different *component images* screened at the same position but at three different wavelengths.

The whole data set of *component images* screened from an *array plate* is called an *array image*. In some cases, an *array image* is combined to a large composite *montage image*. Figure 1.4 shows an example of an *array image* generated from screening a whole *array plate* for three different wavelengths.

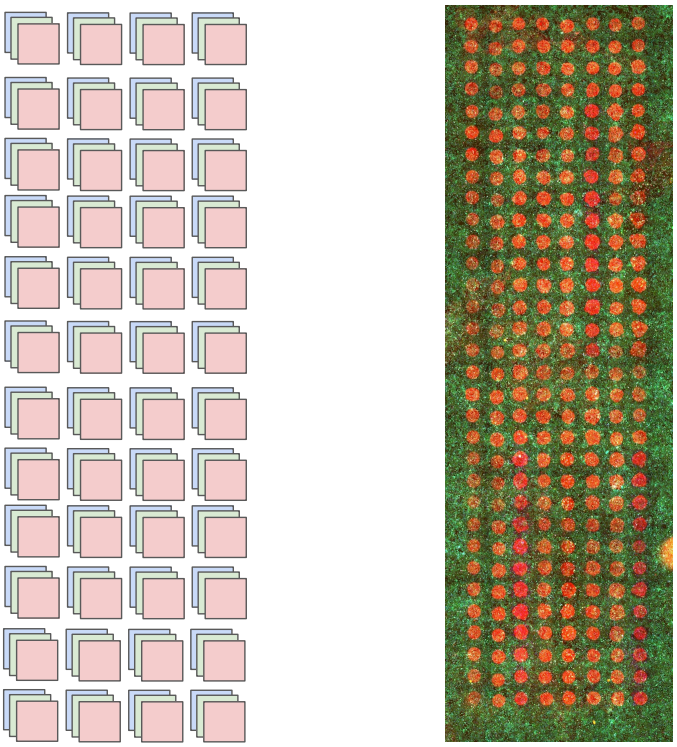


Figure 1.4: To the left is an example of an *array image* that consist of $(3 \cdot 4 \cdot 12 = 144$ *component images* generated from screening a whole *array plate* for three different wavelengths. To the right is a 3-channel composite *montage image* generated from the *component images* that in total contains 256 experiment spots.

In order to apply further quantitative analysis on each individual experiment (within each red spot), individual *spot images* has to be generated. In Figure 1.5, a *spot image* is shown. To generate those images, a reliable, accurate and efficient spot segmentation is required.

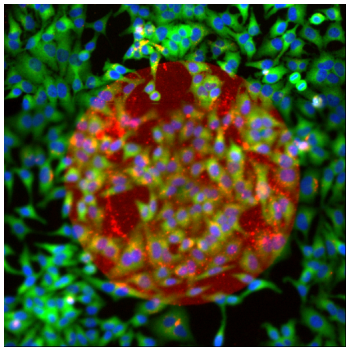


Figure 1.5: An enhanced 3-channel *spot image*.

1.1.3 The Dataset

The dataset used in this thesis is provided by Persomics [1] and consists of 68 screened *array images* that holds 26660 experiment spots, where nine of the *array images* are double-sided. Each *component image* has *tiff* file format and 16-bits per pixel. The total amount of *component images* and the total disk size in this dataset is approximately 24 000 files and 225GB respectively. A detailed summary of the dataset is presented in Table 1.2.

Print number	Rows	Columns	Spots	Count	Total Spots
p67	32	8	256	2	512
p68	32	8	256	5	1280
p69	32	8	256	2	512
p70	32	10	320	8	2560
p71	32	7	224	2	448
p74	16	12	192	16	3072
p76	32	5	160	2	320
p79	32	2	64	1	64
p80	16	10	160	5	800
p81	32	2	64	5	320
p82	32	12	384	2	768
p82-d	32	24	768	4	3072
p83	32	7	224	1	224
p84	32	19	608	5	3040
p84-d	32	38	1216	5	6080
p85	32	38	1216	3	3648
Sum				68	26720

Table 1.2: A detailed summary of the image dataset. Count is the number of *array images* of respectively *print number* and the double-sided *print types* are denoted with a 'd' in the *print numbers*' name.

The dataset comes with spots and image data of varying quality which is visualized in Figure 1.6

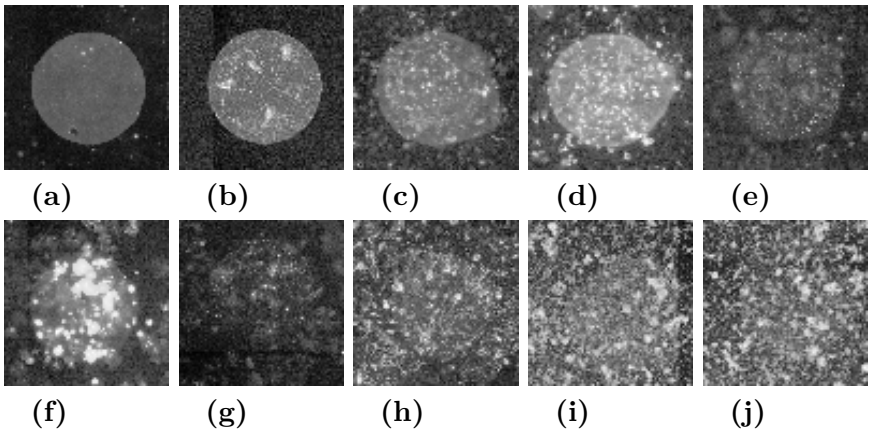


Figure 1.6: Different spot qualities and visibilities in descending order from (a) to (j), where (a) is considered easiest to detect and (j) the hardest. The most common qualities are in the range (c) to (g) for the whole dataset.

1.2 Aim

The master's thesis aim is a developed program that successfully detects all individual spots and maps them to each corresponding experiment from the real fluorescent microscope images in the provided dataset. The program should handle *component images* as input, not rely on annotation coordinate data, handle disoriented *array images* and displaced spots and efficiently process datasets of a very large size.

1.3 Limitations

This thesis will not deal with the issue when non-standard naming conventions of *component images* are used, i.e, when the row, column and wavelength order is completely unknown by analyzing file names. An assumption is also made that there are no under- or overlapping regions between *component images*.
Moreover, a solution for disoriented *array images* is not covered in this study since it is not possible to detect the orientation of single-sided *print types*.

1.4 Specification of Issue under Investigation

In this section, four research questions specified on the basis of the aim are presented.

RQ 1: What is an effective and robust way of identifying and cropping all the experiment spots into individual spot images? *Accurately detecting spots in fluorescence microscopy images is of primary interest for further quantitative analysis of the experiments such as counting and classification.*

RQ 2: How can disoriented array images and displaced spots be handled? *Each spot is an individual experiment. Knowing which experiment that belongs to a spot is trivial if all screened array images are oriented in the same way, but that can not be guaranteed when users will screen with different imagers. If an array image, for instance, is screened upside-down, it is no longer trivial knowing which experiments that belongs to the spots if the orientation is unknown. Spots can also be displaced due to non perfect production of the array plate. The challenge is mapping the detected spots to each corresponding experiment for disoriented images and displaced spots.*

RQ 3: How can images with different magnification and spots with different radius be handled? *Different microscopy scanners will produce images of varying dimensions and magnifications which means that spots and other features will not have a fixed size. Furthermore, production of the array plates will not be perfect, which means that spots will vary in size on the same array plate. The fundamental challenge here is processing images and spots of unknown size.*

RQ 4: What is an effective way of processing high resolution images of a very large size? *Fluorescence microscopy scanners are capable of producing images of sizes up to several tens of gigabytes each. Complex image processing operations applied on images of a very large size is not only a bottleneck for spot detection, but also for efficient integration of extensive imaging data in general research.*

1.5 Related Work

To this study, there are a lot of related work. The first subsection introduces the currently used detection framework developed by Persomics. The following subsection describes others' work that is related to the study in this thesis.

1.5.1 Current Detection Framework

The detection framework developed by Persomics can be summarized by following steps: 1) use all image data in the red channel in a montage image and decimate it into a smaller one, 2) enhance the spots with histogram equalization, 3) remove noise with a median filter, 4) threshold the resulting image with Otsu's method, 5) compute connected components, 6) remove component outliers, 7) calculate a bounding rectangle box around the components and 8) calculate the spot positions by transforming annotation coordinates from a database to cover the whole bounding box. More details about this framework can be found at [1].
The greatest drawback with the current framework is that it rely on saved annotation coordinates for each *print number*. To generate annotation data for a new print requires many man-hours of work which is costly and inefficient. Other drawbacks is that it can not handle *component images* as input as well as being sensitive to noise, artifacts and contamination which is common in microscopy images. The framework also heavily rely on the detected corner spots to generate the bounding box: if any is missing or their centers are detected slightly wrong, the whole spot detection will fail.

1.5.2 Relevant Research Articles and Literature Review Directions

Beyond the current detection framework, there are a lot of related work to this study. In [4] is a dozen of different spot detection methods provided with a thorough evaluation of each method. Smal et al [4] is explaining that it is common to first de-noise the image and enhance the signal (the spots) as image preprocessing framework. Then, the minimum or maximum values of the spots are extracted. The easiest method of spot detection in a gray-scale image is calculating the intensity histogram and then thresholding the image intensities. By using methods such as entropy minimization [5] or Otsu's approach [6], automatic selection of the threshold value can be carried out.
The images segmentation approaches in this thesis uses Otsu's method and a simplified version of the similarity search that is used in [7]. In [7], segmentation is achieved by sampling the input image into tiles and searching for similar tiles from a database. The retrieved tiles are processed with different descriptors that describes specific image characteristics and multiple dissimilarity values are combined for each descriptor into a segmented image. The method in this thesis does not process the tiles, it uses the distances between tiles as dissimilarity values. One drawback with both methods are that they have a relatively high computational complexity. Beyond similarity search, there are also other candidates for image segmentation such

as using haar Wavelet transform for feature extraction as in [8] or convolutional neural networks as in [9]. Compared to these methods, neural networks are very expensive to train, but once trained, it is relatively cheap to segment images, which is an advantage [10].
When the image data is segmented, a common method to detect spots is gridding. In [11], an approach for finding spot locations is proposed which uses a hill-climbing method to maximize the energy of the spots and fit it to different probabilistic models. Another gridding approach is mathematical morphology where the images are represented as a function and morphological filters and erosion operators are applied, which removes ridges and peaks from the image topological surface [12]. Furthermore, a two-stage gridding method has been proposed in [13] where sub-grid locations are found by using optimal multilevel thresholding and the locations of spots are found within each sub-grid.
In this thesis, a novel approach for spot detection is presented, which is based on the two-stage gridding method, but instead of using optimal multilevel thresholding, it generates its grid from detected peaks and valleys in the intensity profiles of segmented images.

1.6 Outline

This thesis is structured as follows. The second chapter introduce the theoretical foundations from the field of image processing. The third, forth and fifth chapters describes the methodology used in the data structuring, image pre-processing and spot detection frameworks respectively. The sixth chapter contains a detailed evaluation of the whole spot detection framework. Finally, the results and future work are discussed in chapter seven.

2
Theory

In this chapter, the theoretical foundations of digital imaging, image enhancement and image segmentation are presented.

2.1 Image Representation

An image can mathematically be defined as a two-dimensional function, $I(x\,,y)$, where x and y are pixel coordinates and the amplitude of I at any pixel coordinate $(x\,,y)$ is called the *gray level* or *intensity* of the image at that pixel. When the pixel coordinates and the intensity values are all discrete quantities, the image is called a digital image [14, p. 23].
In image processing, it is important to pay attention to different coordinate system conventions. The center of the upper-left pixel in an image is often considered as the origin $I(0\,,0)$. There is more variation in how the x and y axes are assigned. The coordinate system in Figure 2.1, where the x-axis is pointing down and the y-axis is pointing to the right is used in order to be consistent with the representation in [14].

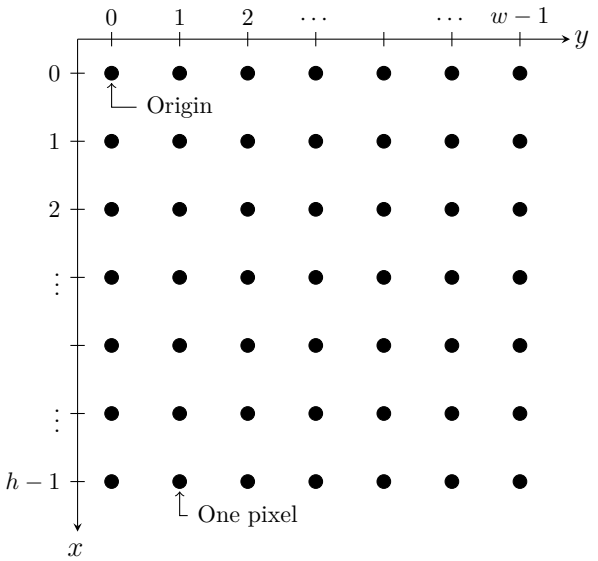


Figure 2.1: The chosen image coordinate convention where the x-axis is pointing down, the y-axis is pointing to the right and origin is defined as the upper-left pixel.

Digital images can be represented as matrices or arrays, which happens to be great

for working with in Python. In equation form, a monochrome digital image can be represented as an h x w numerical array as

$$I(x\,,y) = \begin{bmatrix} I(0\,,0) & I(0\,,1) & \cdots & I(0\,,w-1) \\ I(1\,,0) & I(1\,,1) & \cdots & I(1\,,w-1) \\ \vdots & \vdots & & \vdots \\ I(h-1\,,0) & I(h-1\,,1) & \cdots & I(h-1\,,w-1) \end{bmatrix}, \tag{2.1}$$

where h is the image height and w is the image width. A color image can be given by c different monochrome (gray-scale) images in a h x w x c numerical multidimensional array, each of them representing one dimension of the color data, where each resulting image is denoted a color *channel*.

2.2 Image Enhancement

Image enhancement techniques are used for many different applications of image processing. The methods are used to alter a digital image in order to enhance certain aspects of it so the results are more suitable for further image analysis steps such as spot segmentation. Examples of basic operations that can be applied on a gray-scale image is multiplication and addition of the intensity values. Negative addition would result in a darker image as each intensity value would decrease and positive addition would result in a brighter image. Multiplication of a value > 1 increases the contrast (the range of the intensity values) whilst multiplication with a value $\in (0\,,1]$ results in an image with lower contrast. Those examples of operations can be done pixel by pixel where each pixel's change is totally independent of the other pixels. There are also other operations which not is independent of the other pixels values, which are described in following sub-sections where histogram equalization is introduced.

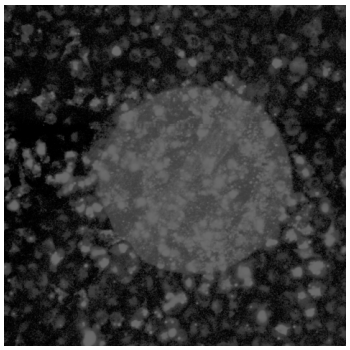
2.2.1 Histogram

The histogram of an image can be seen as a graph with the frequency of every intensity value in the image. For a digital image with intensity levels $r \in [0\,,L-1]$, the histogram is defined as a discrete function $h(r_k) = n_k$, where r_k is the k^{th} intensity value and n_k is the number of pixels in the image with intensity r_k. A histogram can thus be seen as a representation of an image where the pixel positions are lost, but instead the amount of each possible intensity in the image is made accessible for analysis [14, p. 142]. Histograms have several possible applications in the fields of image segmentation and image processing. One of these will be presented in the following sub-section, namely histogram equalization.

2.2.1.1 Histogram Equalization

Histogram equalization is a method that is usually useful in images with details of interest that are either under or over-exposed. The method stretches the intensity range in an image so the histogram is mapped into a more uniformly distributed one, which results in an image where the overall contrast is improved and the intensity

(a) Original image.



(b) Image processed with histogram equalization.

Figure 2.2: Histogram equalization applied on an example image.

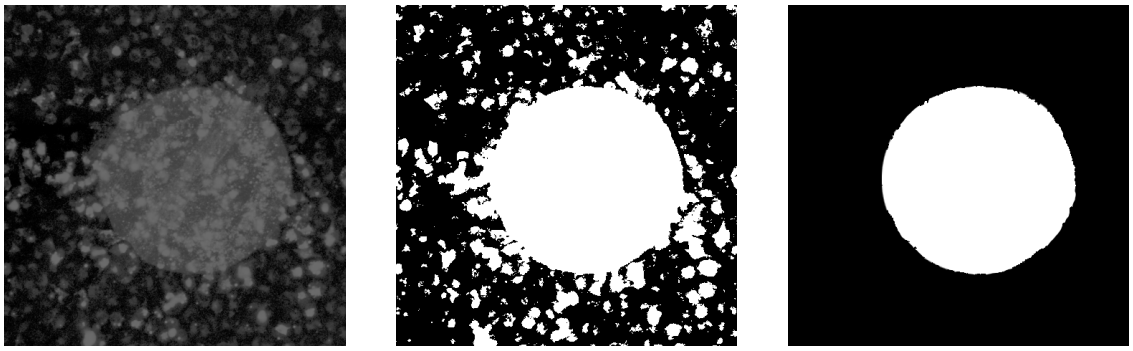
values are spreaded over the whole range. This equalization effect is achieved using the the cumulative distribution function (*cdf*) as remapping (more details is provided in [14, p. 144]). The effects of histogram equalizing a low-contrast *spot image* can be seen in Figure 2.2b.

2.3 Image Segmentation

A digital image given by a set of pixels is usually considered as low-level information, the pixels only specifies their relative position and the local intensity of a scene. The aim of image segmentation is to extract high-level information (information about objects in an image) and alter the image representation into something more meaningful. Since it is rarely possible to tell much about a scene by analyzing a single pixel, a common approach is to group pixels into larger sets, called *segments*. One example of this can be seen in Figure 2.3a, where an example image is segmented into two pixel groups as in Figure 2.3c. One of the segmentation approaches used in this thesis builds on a technique called thresholding.

2.3.1 Optimum Global Thresholding Using Otsu's Method

Thresholding is an image segmentation method used for separating an image into two or more groups (*classes*) where each class has a certain intensity level. Otsu's thresholding method is optimum in such way that it maximizes the between-class variance and is based on the idea that the classes in a well-thresholded image should be unambiguous with respect to their pixel intensity values [14, p. 764]. The approach includes iteration through all the feasible threshold values and computation of a measure of spread for the pixel intensities for each side of the threshold. Its implementation and methodology is described more thoroughly in [14, p. 765] and an example of output is visualized in Figure 2.3b, where the foreground and background pixels are segmented.



(a) Original image.

(b) Image processed with Otsu's method.

(c) Manually defined ground-truth for image segmentation.

Figure 2.3: Image segmentation with Otsu's method on an example image, compared with ground truth.

3

Image Metadata Structuring

From a single *array plate* screening, thousands of *component images* can be produced with a total data size of several tens of gigabytes. Instead of generating a huge *montage image* from all *component images*, it would be much more efficient and satisfactory to come up with a framework that makes it possible to treat the *component images* as if they were montaged. In this chapter, a framework for regex parsing and a framework for data structuring and reading from the data structure is presented.

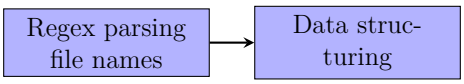


Figure 3.1: Overview flowchart of the metadata structuring framework.

3.1 Regex Parsing

Each *component image* from Persomics have multiple metadata of interest in its file name, namely row and column position, channel, name and file name extension, where channel, row and column are crucial information in order to read global image data in an *array image*. This metadata can be seen in the following example file name from Persomics where each metadata group is identified:

Name: Persomics 10X U2OS 72h p75_ | Row: O | Column: 13 | _w | Channel: 3 | .TIF: Type

In order to acquire the substrings for each metadata group in a target string, a *regular expression* can be used, which is a codified method of searching [15]. There is useful terminology related to regular expressions defined in [15] which can be used, namely;

Literal
: Any character used in a search or matching expression. It is the string that is literally being searched for.

Metacharacter
: One or more special characters that have a unique meaning. They are not used as literals in the search expression.

Target string
: The string that should be parsed.

Escape sequence
\(backslash) is used before one metacharacter to indicate that it should be used as a literal.

As stated in [16], multiple primitive regular expressions can be merged into a new, more complex regular expression. By using this method, the example target string can found by eight primitive regular expressions, one defining 'Name', first literal, 'Row', 'Column', second literal, 'Channel', third literal and 'Type', where the three literals are '_', '_w' and ' . ' respectively. By using the methodology and some of the metacharacters defined in [16], eight regular expressions can be combined together into a final regular expression as in Table 3.1.

File convention	Regular expression
Persomics	(?P<Name>.*)_(?P<Row>[A-Z]{1,2})(?P<Col>[0-9]{2,3}) _w(?P<Channel>[0-9])\.(?P<Type>[A-Z]{3})

Table 3.1: A regular expression for parsing file names from Persomics.

3.2 Data Structuring

When a regular expression is applied to a target string, it is satisfactory to store the metadata output in a data structure. To have an example to work with, the regular expression for Persomics in Table 3.1 is applied to 12 different file names and the result can be seen in Table 3.2 for the groups Row, Column and Channel.

File #	File name	Row	Column	Channel
0	'Persomics 10X U2OS 72h p75_E11_w2.TIF'	E	11	2
1	'Persomics 10X U2OS 72h p75_B10_w1.TIF'	B	10	1
2	'Persomics 10X U2OS 72h p75_E10_w1.TIF'	E	10	1
3	'Persomics 10X U2OS 72h p75_C10_w2.TIF'	C	10	2
4	'Persomics 10X U2OS 72h p75_B10_w1.TIF'	B	10	1
5	'Persomics 10X U2OS 72h p75_E10_w1.TIF'	E	10	1
6	'Persomics 10X U2OS 72h p75_E11_w2.TIF'	E	11	2
7	'Persomics 10X U2OS 72h p75_B11_w2.TIF'	B	11	2
8	'Persomics 10X U2OS 72h p75_C11_w1.TIF'	C	11	1
9	'Persomics 10X U2OS 72h p75_B11_w2.TIF'	B	11	2
10	'Persomics 10X U2OS 72h p75_C11_w1.TIF'	C	11	1
11	'Persomics 10X U2OS 72h p75_C10_w2.TIF'	C	10	2

Table 3.2: Some examples of *component image* file names from Persomics.

The metadata is chosen to be stored in the data structures *list* [17] and *class* [18] since they are easy to implement in Python. Let's denote the list of rows as R, the list of columns as C and the list of channels as W from a regex parsing output. The contents of these lists can be stored in a component image class. To make things easier in Python where element indices starts at zero, it is satisfactory to convert the content in each list to indices starting at 0 and upwards. An example is the row list R = [E, B, E, C, B, E, E, B, C, B, C, C] which can be replaced with the indices

[2, 0, 2, 1, 0, 2, 2, 0, 1, 0, 1, 1]. To achieve such output, the lists can be processed as described in the next section.

3.2.1 Data Replacement in Regex Output

It it satisfactory to replace the contents in the lists R, C and W with indices (starting at 0 and upwards). To replace the list contents, a first step can be to take the set of each list, denoted $\{.\}$, where the set operator returns an unordered collection with no duplicate elements [19] as the following example:

$$\{1\,,2\,,4\,,4\,,4\} = \{1\,,2\,,4\}\,. \tag{3.1}$$

By using the set operator on the lists R, C and W, the output is $\{R\}$ = {E, B, C}, $\{C\}$ = {11, 10} and $\{W\}$ = {2, 1}. After the set operation is applied, the output can be sorted in ascending order. Let's denote this sorting operator as *sort*(). By applying this operator on each set, the results are *sort*($\{R\}$) = {B, C, E}, *sort*($\{C\}$) = {10, 11} and *sort*($\{W\}$) = {1, 2}. From each sorted set, an index dictionary $D = \{\{v_0\,, i_0 = 0\}\,, \ldots\,, \{v_n\,, i_n = n\}\}$ can be generated where the first element v_j is the value in the sorted set, the second element i_j is the index of the value in the sorted set, n is the length of each sorted set and j is the range from 0 to n. The index dictionaries of the sorted sets are generated as D_R = {{B, 0}, {C, 1}, {E, 2}}, D_C = {{10, 0}, {11, 1}} and D_W = {{1, 0}, {2, 1}}. The elements in the lists R, C and W can finally be replaced by using the corresponding index dictionary, where each list element which is equal to any of the first dictionary values v_j is replaced with corresponding index i_j.

3.2.2 Read Data from Data Structure

When the file names of *component images* are parsed with regular expression, information about their location and channel are extracted, which is crucial information in order to read image data from a certain region in the global *array image*. A "toy" example of an *array image* is illustrated in Figure 3.2 consisting of 8x8 = 64 *component images* $I^{\mathrm{c}}_{i,j}$ in rows $i = \{0\,, 1\,, \ldots\,, 7\}$, columns $j = \{0\,, 1\,, \ldots\,, 7\}$ with a sub-region that is of interest for extraction. Lets call this extracted sub-region as the *region of interest image*, denoted I^{roi}. This toy example can be used to come up with a general method to extract image data from any set of *component images* in an *array image*.
To keep track of indices, three different indexing systems can be implemented, one for the *array image*, one for a local *component image* and one for the region of interest image. The global indices are denoted with large letters, $X\,, Y\,$, and the local indices in a *component image* $x\,, y\,$, and $x_{\mathrm{roi}}\,, y_{\mathrm{roi}}$ for the region of interest image. The height and width of the *component images* are defined as h and w respectively. In the example image, the global starting indices are $(X_s\,, Y_s) = (2.5h\,, 0.5w)$ and the ending indices $(X_e\,, Y_e) = (7.5h\,, 3.5w)$. As can be seen in Figure 3.2, the sub-region starts in the center of the *component image* in row two, column zero and ends in the center of row six, column three. The starting $(r_s\,, c_s)$ and ending $(r_e\,, c_e)$ indices can be calculated as

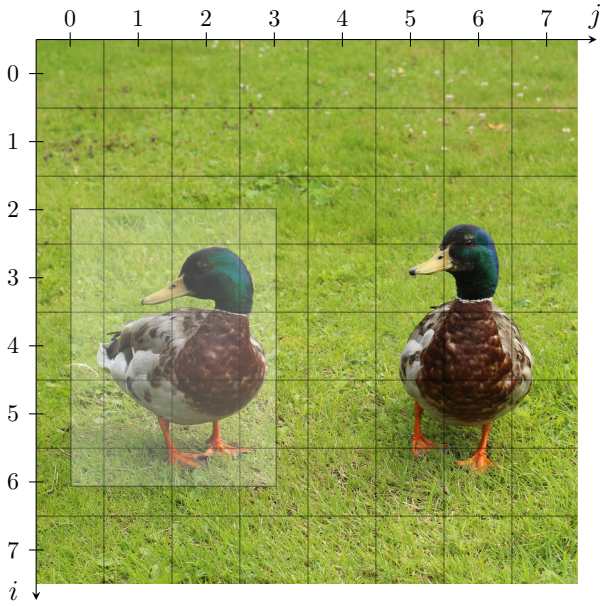


Figure 3.2: An *array image* consisting of multiple *component images* in rows i and columns j with a sub-region that should be extracted, highlighted in white.

$$r_s = \left\lfloor \frac{X_s}{h} \right\rfloor, \quad r_e = \left\lfloor \frac{X_e}{h} \right\rfloor, \quad c_s = \left\lfloor \frac{Y_s}{w} \right\rfloor, \quad c_e = \left\lfloor \frac{Y_e}{w} \right\rfloor, \tag{3.2}$$

where row is denoted r and column c, and $\lfloor . \rfloor$ is the floor operator, which returns the largest integer less than or equal to a given number [20]. All *component images* that have to be read are those who have a row and column index within the ranges

$$R = [r_s \, , r_e] \, , \quad C = [c_s \, , c_e] \, , \tag{3.3}$$

where R are all the row indices and C are all the column indices.
The starting $(x_s \, , y_s)$ and ending indices $(x_e \, , y_e)$ that spans the region of interest within each *component image* in R and C can be calculated as

$$\begin{aligned}
x_s(i) &= \begin{cases} X_s - R[i] \cdot h, & \text{if } X_s > R[i] \cdot h \\ 0, & \text{otherwise} \, , \end{cases} \\
x_e(i) &= \begin{cases} h, & \text{if } X_e > (R[i] + 1) \cdot h \\ X_e - R[i] \cdot h, & \text{otherwise} \, , \end{cases} \\
y_s(j) &= \begin{cases} Y_s - C[j] \cdot w, & \text{if } Y_s > C[j] \cdot w \\ 0, & \text{otherwise} \, , \end{cases} \\
y_e(j) &= \begin{cases} w, & \text{if } Y_e > (C[j] + 1) \cdot w \\ Y_e - C[j] \cdot w, & \text{otherwise} \, . \end{cases}
\end{aligned} \tag{3.4}$$

By using equation (3.4) on the example in Figure 3.2 for $i = 0 \, , \, j = 0$, i.e, $R[0]$ and $C[0]$ (the most top left *component image*), the starting $(x_s \, , y_s)$ and ending indices $(x_e \, , y_e)$ are calculated as

$$
\begin{aligned}
x_s(0) &= 2.5h - 2 \cdot h = 0.5h\,, \\
x_e(0) &= 7.5h - 2 \cdot h = 3.5h \Rightarrow x_e(0) = h\,, \\
y_s(0) &= 0.5w - 0 \cdot w = 0.5w\,, \\
y_e(0) &= 3.5w - 0 \cdot w = 3.5w \Rightarrow y_e(0) = w\,,
\end{aligned}
$$

i.e, the ranges with the data of interest are $x = [0.5h\,, h]$ and $y = [0.5w\,, w]$ inside the *component image* in $R[0]$ and $C[0]$.
Finally, the starting $(x_{\text{roi}\,,s}\,, y_{\text{roi}\,,s})$ and ending indices $(x_{\text{roi}\,,e}\,, y_{\text{roi}\,,e})$ in the region of interest image are calculated as

$$
\begin{aligned}
x_{\text{roi},s}(i) &= \begin{cases} x_{\text{roi},s}(i-1) + x_e(i) - x_s(i), & \text{if } i > 0 \\ 0, & \text{otherwise}\,, \end{cases} \\
y_{\text{roi},s}(j) &= \begin{cases} y_{\text{roi},s}(j-1) + y_e(j) - y_s(j), & \text{if } j > 0 \\ 0, & \text{otherwise}\,, \end{cases} \\
x_{\text{roi},e}(i) &= x_{\text{roi},s}(i) + x_e(i) - x_s(i)\,, \\
y_{\text{roi},e}(j) &= y_{\text{roi},s}(j) + y_e(j) - y_s(j)\,.
\end{aligned}
\tag{3.5}
$$

Those indices corresponds to the region inside the region of interest image that maps the data extracted from the *component image*. By using Equation 3.5 on the example in Figure 3.2 for the most top left *component image* in $R[0]$ and $C[0]$, the region of interest indices can be calculated as

$$
\begin{aligned}
x_{\text{roi},s}(0) &= 0\,, \\
y_{\text{roi},s}(0) &= 0\,, \\
x_{\text{roi},e}(0) &= 0 + x_e(0) - x_s(0) = 0.5h\,, \\
y_{\text{roi},e}(0) &= 0 + y_e(0) - y_s(0) = 0.5w\,.
\end{aligned}
$$

To extract all data into the region of interest image, this process has to be repeated for all *component images* in R and C as

$$
\begin{aligned}
I^{\text{roi}}(x_{\text{roi},s}(i) : x_{\text{roi},e}(i), y_{\text{roi},s}(j) : x_{\text{roi},e}(j)) = I^c_{i,j}(x_s(i) : x_e(i), y_s(j) : y_e(j))\,, \\
\text{for } i \in R\,, \quad \text{for } j \in C\,,
\end{aligned}
\tag{3.6}
$$

which can be implemented in Python as in Appendix A.2.

4

Image Preprocessing

The input to the image preprocessing framework are *red-channel component images* read from the data structure in the previous chapter. The reason why only data from the red channel is chosen is since they are screened at the wavelength where the red dye in the spots emits light. The *component images* are downsampled and montaged into a composite image. The output is an enhanced, quantized composite image.

Figure 4.1: Overview flowchart of the image preprocessing framework.

4.1 Image Decimation

Fluorescence microscopy scanners are capable of producing image datasets of sizes up to several tens of gigabytes each and complex image processing operations is a bottleneck for spot detection. In order to reduce the computational complexity, one efficient way is to decrease the amount of samples by using *image decimation* and perform spot detection in the downsampled data.

4.1.1 Decimation Factor

In image decimation, the decimation factor M used for downsampling is the ratio of the input rate to the output rate. It means using M keeps only every Mth sample of the input [21]. When image decimation is used, it should be noted that the uncertainty where spots are located increases with the decimation rate. The fundamental tradeoff in the choice of decimation rate is the resulting reduced computational complexity versus the amount of data needed to accurately represent the locations of spots. The decimation rate is empirically determined as the integer value closest to 10% of the average spot radius in pixels for the provided dataset.

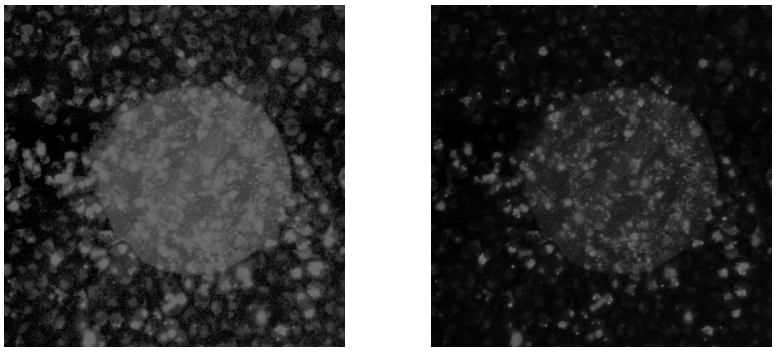
4.1.2 Decimation Algorithms

An image can be downsampled with several different approaches. From a computational standpoint, the easiest decimation algorithm to implement is nearest neighbor interpolation, where each pixel is given the value of the sample which is closest to

it [22]. Nearest neighbor interpolation causes high-frequency signal components to be misinterpreted, which results in an image with undesirable jaggedness which is a form of distortion called *aliasing*. An example of jaggies are diagonal lines, which shows a "stairway" shape in the nearest neighbor interpolated image [23]. There are other decimation methods that can be used which suppress the aliasing to an acceptable level. In [24], different image interpolation algorithms are compared for speed and quality after downsizing. From this comparison, it can be concluded that *pixel area relation* is not the fastest method, but it yields the best image quality and avoids aliasing. Pixel area relation is therefore chosen as decimation method for the *component images*.

4.2 Image Enhancement

To enhance the spot signal, *histogram equalization* can be used as described in Section 2.2.1.1. As stated by Richard Kirk which is a contributor to Image J's contrast enhancer plug-in, histogram equalization can in some cases enhance meaningless detail and hide important but small high-contrast features [25]. The proposed approach is to use his method, which is a modified version of the histogram equalization algorithm where the square root is applied on the histogram values, resulting in effects that are less extreme. The output differences between the two methods are visualized for an example image in Figure 4.2.



(a) Histogram equalization

(b) Richard Kirk's histogram equalization

Figure 4.2: Histogram equalization and Richard Kirk's histogram equalization applied on an example image.

4.3 Image Quantization

To reduce the computational complexity for spot detection, *quantization* can be used. When an image is quantized, pixel amplitude values are replaced with approximate values taken from a finite set of allowed values [26]. One of the basic choices in quantization is the number of discrete quantization levels to use. The fundamental tradeoff in this choice is the resulting signal quality versus the amount of data needed to represent each sample. A simple quantizer that can be used is

the *uniform quantizer*, which is described thoroughly in [26], where the transition and reconstruction levels are all equally spaced. Figure 4.3 show an example of the effects of reducing the number of bits to represent the gray levels in an image by using uniform quantization.

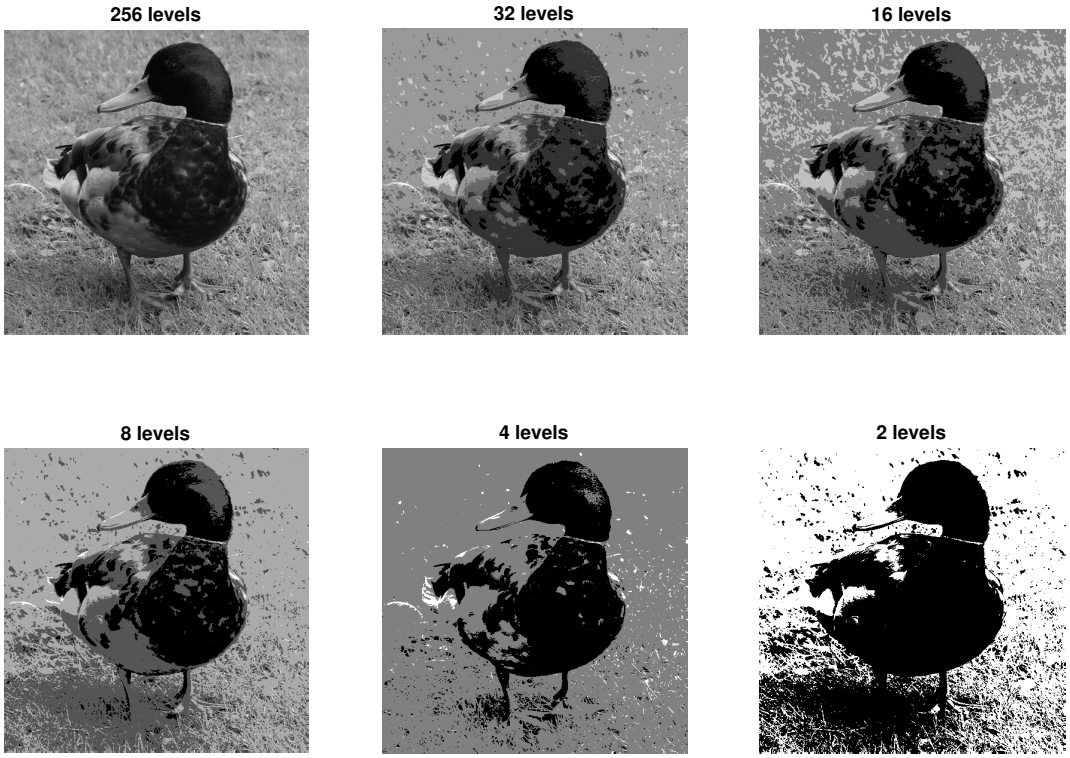


Figure 4.3: Quantizing a gray-scale image with different quantization levels by using uniform quantization.

The chosen method is to decrease from 65535 gray levels (16-bits per pixel) to 256 gray levels (8-bits per pixel) by using uniform quantization. 8-bits are usually sufficient to represent the intensity.

5

Spot Detection

The input to the spot detection framework is the processed composite montage image from the previous chapter. The composite image is first segmented with a k-NN classifier, generating a classification map with estimates of where spots are located. If the *print type* of the *array image* is double-sided, the sub-array regions are detected. A global grid is generated by grid fitting and spots are finally detected within each grid region.

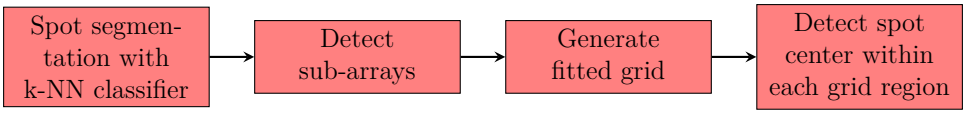


Figure 5.1: Overview flowchart of the spot detection framework.

5.1 Spot Segmentation with k-NN

The proposed spot segmentation approach is based on the methodology in [7]. The input to the spot segmentation algorithm is the decimated histogram equalized montage image from the image processing framework in the previous chapter. The output of the algorithm is a *classification map* with the same size as the input. The classification map contains, for each pixel, a likelihood that the pixel belongs to a spot. In Figure 5.2, the classification map is visualized with white colors and the likelihood is represented with their brightness. The darker the color, the lower is the likelihood that the pixel contains spot. The spot segmentation can be divided into the following steps:

1. The input image is sampled into smaller square regions (called tiles).
2. For each sampled tile, the most similar tiles of known spots and background are retrieved from a *training database* by using k-NN query. The retrieved data is processed and the likelihood that the tile from input image contains spot is estimated.
3. The classification result of each tile is combined into a classification map.
4. The classification map is up-sampled to the same size as the input montage image.

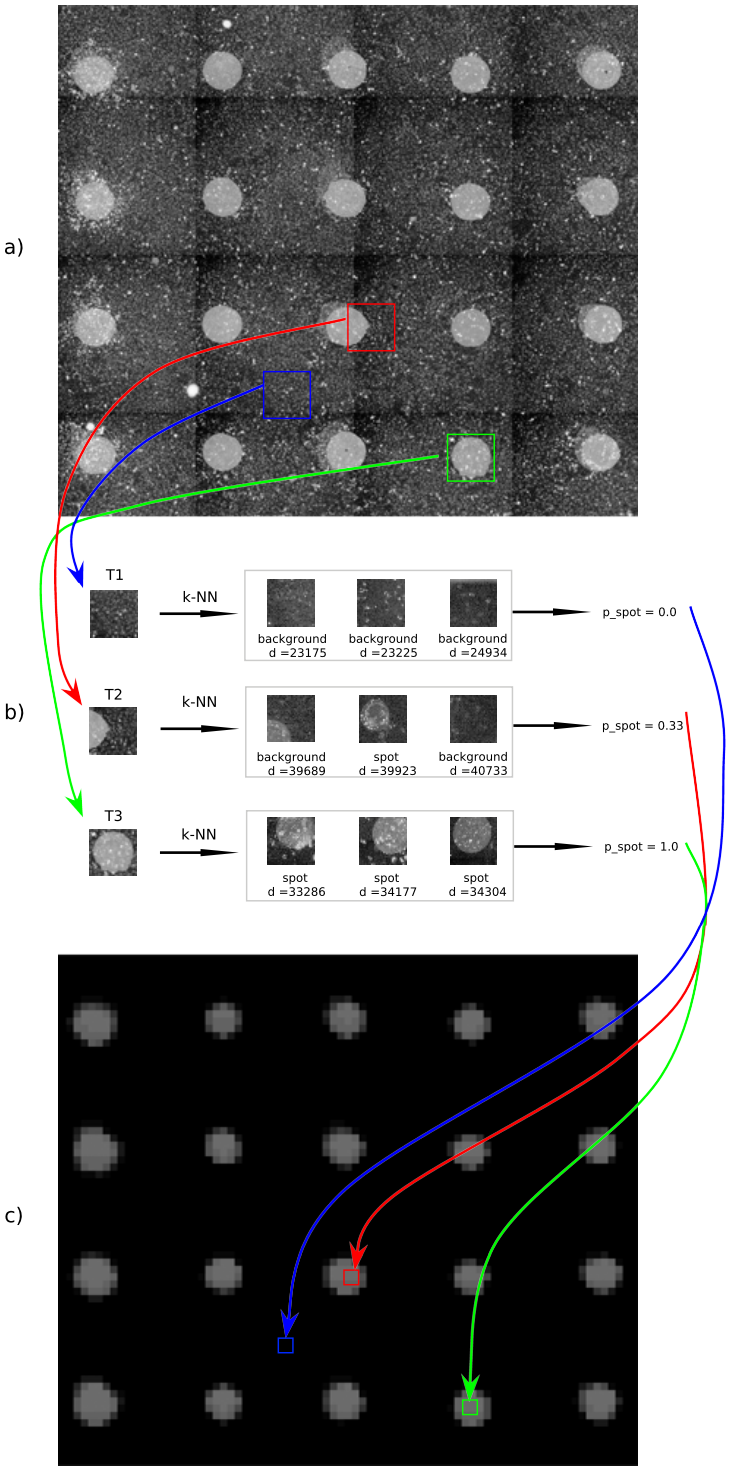


Figure 5.2: Illustration of the spot segmentation procedure. a) Tiles T_1, T_2 and T_3 are generated from the input image. b) The most similar tiles are retrieved from the training database for each sample tile by using k-NN query ($k = 3$). The likelihood that the sample tile contains spot is estimated and the results of each tile is combined. c) Classification map, which contains, for each pixel, the likelihood that the pixel belongs to a spot. The higher intensity, the more likely.

5.1.1 Sampling of Input Image

The input image is divided into regular square tile regions with an overlap between the tiles. The tile size 32 × 32 px with a step size $s = 10$px, resulting in an overlap of 22 px is chosen. Moreover, a *zero padding* $z = 16$px (half the tile size) is used for straightforward implementation of the upsampling process. The sampling procedure is illustrated in Figure 5.3.

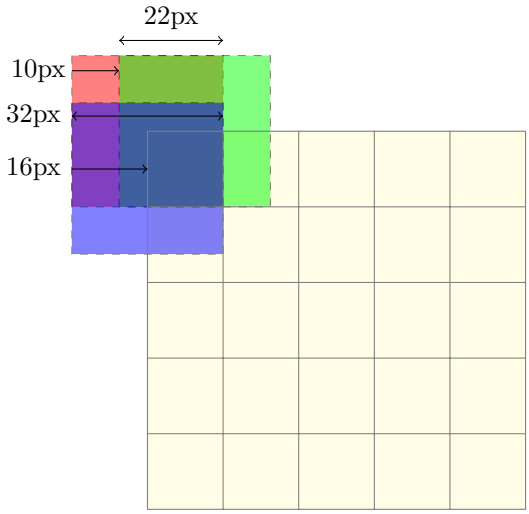


Figure 5.3: Segmentation of the input image into tiles of size 32 x 32px. An overlap of 22px is used which is achieved by moving the region with a step size $s = 10$px. A zero-padding $z = 16$px (half the tile size) is used.

By using this sampling approach, it can happen that a tile may contain both spot and background regions. As stated in [7], it should not be an issue since the similarities of the retrieved tiles from the database are combined together. Overlaps are used in order to reduce uncertainties in the resulting classification map. The tile size is determined empirically in such way that they contain enough characteristic features for classification of spots and background. A too small sample would tend to not contain enough visual features and the amount of samples would be rather high. A too large tile would contain more than one spot or too many different features, which could decrease the classification performance.

5.1.2 Classification with k-NN

For each generated tile, the k most similar training images are retrieved from the training database by using k-NN query as in [7]. Let T_i denote the i^{th} tile from the image sampling. The output from the k-NN$(T_i\,, k)$ query contains k dictionaries $\{D_1^i\,, D_2^i\,, \ldots\,, D_k^i\}$. Each output dictionary D_j^i can be written in a form of a set $D_j^i = \{t_j^i\,, d_j^i\,, c_j^i\}$, where t_j^i represents the training image from the training database, d_j^i denotes distance to the sampled tile image T_i and $c_j^i \in [0\,, 1]$ is the class to which the sample t_j^i belongs, where 0 is background and 1 is spot. The distance can be calculated as

$$d_j^i(T_i\,,t_j^i) = \sum_{x=0}^{h}\sum_{y=0}^{w} |T_i(x\,,y) - t_j^i(x\,,y)|\,, \tag{5.1}$$

where h and w are the tile image height and width [10]. A simple example using pixel-wise differences is given in Figure 5.4.

$$\left| \underbrace{\begin{pmatrix} 255 & 32 & 10 \\ 90 & 23 & 128 \\ 24 & 26 & 178 \end{pmatrix}}_{\text{tile image } T} - \underbrace{\begin{pmatrix} 10 & 20 & 24 \\ 8 & 10 & 89 \\ 12 & 16 & 178 \end{pmatrix}}_{\text{training image } t} \right| = \underbrace{\begin{pmatrix} 245 & 12 & 14 \\ 82 & 13 & 39 \\ 12 & 10 & 0 \end{pmatrix}}_{\text{pixel-wise differences}} \rightarrow \underbrace{437}_{distance}$$

Figure 5.4: A simple example of using pixel-wise differences to compute the distance in Equation 5.1. The two images are subtracted element-wise and all differences are added to a single number.

Based on each output dictionary $\{t_j^i\,, d_j^i\,, c_j^i\}$, the likelihood p_S^i that the sampled tile T_i contains spot is determined as

$$p_S^i = \sum_{j=0}^{k} \frac{c_j^i}{k}\,. \tag{5.2}$$

With this definition, the prediction p_S^i can only have a value $\in [0, 1]$.

5.1.3 Classification Map

When all likelihoods are computed, a classification image can be generated from the likelihood values. Since a step size > 1 is used, the size of the classification map will be smaller than the input image. Up-sampling with nearest-neighbor interpolation is used, which is replacing every pixel with multiple pixels of the same color. The up-sampling factor U is chosen to be equal to the step size s. With this method, the resulting up-sampled classification map will then be of same size as the input image.

5.2 Sub-array Detection

If the screened array is a double-sided array with two sub-arrays, it is satisfactory to separate those sub-arrays and perform spot detection within each sub-array and merge the result together after detection. The proposed approach to detect sub-arrays is visualized in Figure 5.5.
The first step is to threshold the classification image with Otsu's method, generating a binary image. In the binary image, all connected components are computed with 4-connectivity. The components whose width and height ratio is T% larger or smaller than 1 are removed and the centroid for each component that is left is computed. K-means clustering is applied on the centroid coordinates and two clusters

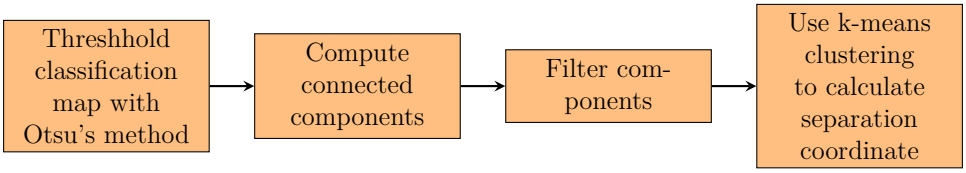


Figure 5.5: Overview flowchart of the sub-array detection.

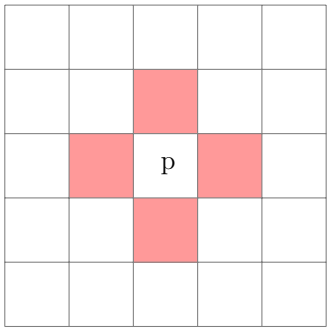
are generated, one for the left and one for the right sub-array. From the k-mean clustering result, the classification image can be cropped into two separated images with one sub-array in each image.

5.2.1 Thresholding

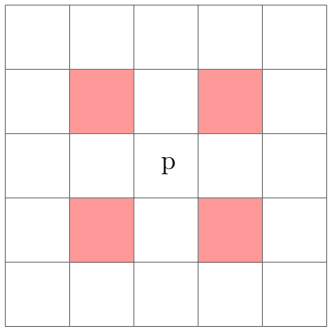
In order to identify the two sub-arrays, the first step in this approach is applying Otsu's thresholding on the classification map, which returns a binary image where the spots are segmented from background. Otsu's method is described in Section 2.3.1.

5.2.2 Connected Components

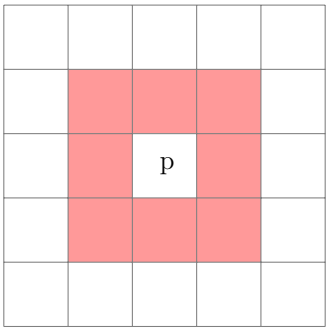
In the thresholded binary image, there are multiple connected regions which are composed of foreground pixels. Those connected regions can be grouped into individual objects, called *connected components* $\{C^0, C^1, \ldots, C^{n-1}, C^n\}$, where each component object C^i holds all pixel coordinates of its *connected pixels*.
Two foreground pixels p and q are said to be *connected pixels* in C^i if there is a sequence of foreground pixels (p_0, p_1, p_n) of C^i where $p_0 = p, p_n = q$ and p_i is a neighbor of p_{i-1} for all $i = 0, 1, \ldots n$ [27, p. 2].
Pixels can be connected in multiple different ways and *pixel connectivity* defines the *pixel neighbourhood*, which is the set of all touching pixels. In Figure 5.6, the effects of different *pixel connectivity* on pixel neighborhood are shown.



(a) 4-connectivity, which is the neighbourhood with only directly touching pixels.



(b) d-connectivity, which is the neighbourhood with the pixels that do not touch it, i.e the diagonal pixels.



(c) 8-connectivity, which is the union of the 4-neighbourhood and the d-neighbourhood.

Figure 5.6: Different pixel connectivity types.

A pretty straightforward method to find connected components is the *recursive algorithm*. It takes a pixel and checks all its neighbours for connectivity. The drawback with this method is that it is inefficient and the computational complexity grows rather quickly with the image size [28]. Details of this algorithm is therefore not covered. A more promising method is the *classical algorithm* designed by Rosenfeld and Pfaltz in 1966. It consists of two passes. In its first pass, it iterates through all pixels. For each current pixel, it checks the pixel to the left and above. By using these two pixel's labels, it labels the current pixel. In the second pass, it refines the result by removing multiple labels for connected regions [28]. The classical algorithm is described more thoroughly in [27, p. 4].
To generate the set of connected component $C = \{C^0, C^1, \ldots, C^{n-1}, C^n\}$ in the thresholded binary image, the proposed method is to use an already developed version of the classical algorithm, found in an Open-CV library [29]. This algorithm is used with 4-connectivity as pixel connectivity.

5.2.3 Filter Components

Among the connected components in C, there might be outliers that are not similar to the size and shape of the experiment spots. A perfect circular spot has a width and height ratio equal to 1. To remove component outliers which definitely not are spots, one way is to filter out all components whose width and height ratio is outside a certain filtering range $R_f = [T_f^{-1}, T_f]$, where $T_f \in [1, \infty)$ is a threshold parameter. By putting a bounding box around each component, the horizontal and vertical size of each component can be computed. The bounding box coordinates of a component C^i can simply be calculated as

$$\begin{aligned} x_1^i &= \min(C_x^i), \\ x_2^i &= \max(C_x^i), \\ y_1^i &= \min(C_y^i), \\ y_2^i &= \max(C_y^i), \end{aligned} \tag{5.3}$$

where C_x^i are all x-coordinates of the components and C_y^i are all y-coordinates of the components. The height and width of a component C^i can be calculated from the bounding box coordinates as

$$\begin{aligned} h^i &= x_2^i - x_1^i, \\ w^i &= y_2^i - y_1^i. \end{aligned} \tag{5.4}$$

5.2.4 Compute Centroids

In computer vision and related fields, image moments are commonly used to describe objects after segmentation. Examples of object properties that can be found with image moments are object area, its total intensity, its centroid and orientation [30]. In this case, image moments can be used in order to compute the centroid coordinate c^i of each connected component C^i.

According to the OpenCV documentation in [31], the spatial moments m_{ji} in a gray-scale image $I(x\,,y)$ can be computed as

$$m_{ij} = \sum_x \sum_y x^i y^j I(x\,,y)\,, \tag{5.5}$$

and the centroids as

$$\bar{x_c} = \frac{m_{10}}{m_{00}}\,, \qquad \bar{y_c} = \frac{m_{01}}{m_{00}}\,. \tag{5.6}$$

The proposed method is to use the pixel data from each connected component as input to equation (5.5) and calculate the centroid coordinates for each component as in equation (5.6).

5.2.5 Separate Sub-Arrays with K-mean Clustering

When the centroid coordinates are computed for the components that are left after filtering, the fundamental challenge is to separate the component centroids into two different groups, one for each sub-array.
One method that can be used is *K-means clustering*, which is a centroid based clustering algorithm. K is an input parameter and represents the number of clusters, which are two in this case. Each centroid among the data of centroids is assigned to a cluster center with the smallest distance to it. There are two key steps in the task of clustering, 1) find the cluster centers and 2), assign each centroid element to the cluster based on its distance. The two steps are iterated until the algorithm converge to a local optimum [32]. The proposed method is to use an already developed K-means clustering algorithm, which is thoroughly described in [33]. When the clusters have been found, a separation coordinate y_s can be calculated, that horizontally separates the sub-arrays in a classification map. The separation coordinate can be calculated as

$$y_s = \max(C_y^1) + \frac{\max(C_y^1) + \max(C_y^2)}{2}\,, \tag{5.7}$$

where C_y^1 are all y-coordinates of the centroids in the first cluster and C_y^2 are all y-coordinates of the centroids in the second cluster.
Figure 5.7 shows an example when K-means clustering is being applied on the centroids found in a classification map.

5.3 Grid Fitting

The input to the proposed grid fitting algorithm is a classification map (or two sub-classification maps from K-means clustering) and the output is a fitted grid with one experiment spot in each sub-grid. The gridding approach is first to calculate the intensity profiles of the classification image. From the intensity profiles, peaks and valleys are detected, where peaks indicates regions where spots are located and valleys indicates regions between spots. The detected valleys are refined by centering

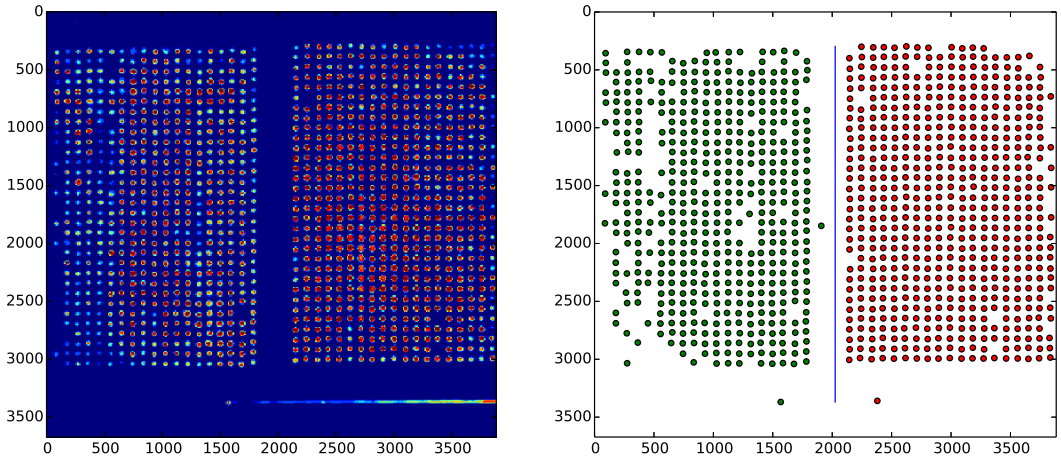


Figure 5.7: The left figure is a classification map of a double-sided array. The right figure is the result of k-means clustering centroids in filtered connected components. The k-mean clustering returns two labeled clusters, where the first cluster is colored green and the second colored red. The blue line is the horizontal separation coordinate where the two clusters are separated and its position is exactly in between the right most centroid in the first cluster and left most centroid in the second cluster.

them between peaks and valley outliers are removed. The final step is to add missing valleys, if any, and generate a global grid from the valley coordinates.

5.3.1 Intensity Profiles

In MATLAB [34], intensity profiles can be used to analyze the intensity values acquired in regularly spaced points along a line segment in an image. This approach can be applied for detection of spot regions by using vertical and horizontal line segments and compute the mean intensity of each line segment. The mean intensity of a horizontal row x and vertical column y in an image can be calculated as

$$\bar{x}(x) = \sum_{y=0}^{w} \frac{I(x\,,y)}{w}\,, \qquad \bar{y}(y) = \sum_{x=0}^{h} \frac{I(x\,,y)}{h}\,, \tag{5.8}$$

where $I(x\,,y)$ is the image, h is the image height and w is the image width. The mean intensity profiles $\bar{X}$ and $\bar{Y}$ of an image are calculated by computing $\bar{x}(x)$ for all $x \in [0\,,h]$ and $\bar{y}(y)$ for all $y \in [0\,,w]$. In Figure 5.8, the mean intensity profiles of an example classification map are plotted.

5.3.2 Peak and Valley Detection

In the classification map image, the spot regions can be extracted from the mean intensity profile arrays $\bar{X}$ and $\bar{Y}$ by detecting peaks and valleys. As stated by Eli in [35], a common peak detection approach is the zero-derivative method, but it can yield false detection due to accidental zero-crossings of the first derivative.

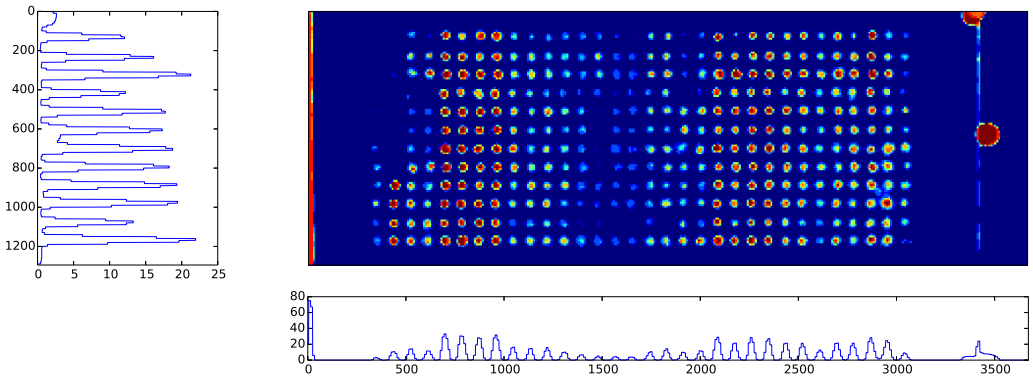


Figure 5.8: Mean intensity profiles calculated over columns and rows in a classification map image.

Another approach is smoothing the input with a low-pass filter, but it usually kills the original signal at the same time. The proposed method is to use Eli's approach for peak and valley detection, which is based on the fact that a peak has lower points around it. In his approach, a point is considered as a peak coordinate (x^+, y^+) if it has the maximal value and was preceded (to the left) by a value lower than a difference threshold Δ. The opposite goes for valleys, where a point is considered as a valley coordinate (x^-, y^-) if it has the minimal value and was preceded by a value higher than Δ. Let's denote the set of detected peak coordinates as $X^+ = \{x_0^+, x_1^+, \ldots, x_n^+\}$, $Y^+ = \{y_0^+, y_1^+, \ldots, y_n^+\}$ and the set of valley coordinates as $X^- = \{x_0^-, x_1^-, \ldots, x_m^-\}$, $Y^- = \{y_0^-, y_1^-, \ldots, y_n^-\}$. By applying the proposed peak and valley detection method on the mean intensity profiles in Figure 5.8, valley coordinates are detected as in Figure 5.9.

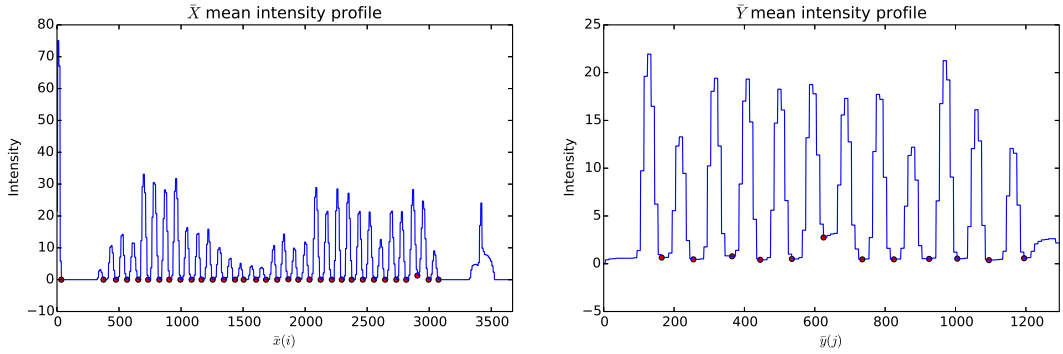


Figure 5.9: Mean intensity profiles calculated over columns and rows in the classification image in Figure 5.8. The local minimums are plotted as red points. As can be seen, an outlier is detected in X^- and the local minimums are not perfectly centered between between the peaks.

As can be seen in Figure 5.9, there is a detected outlier (the left most minimum) in X^-, which has to be removed. Furthermore, the local minimums are not perfectly centered between between the peaks. If the non-centered minimums would be used

for generating the global grid, there is a chance that sub-grids would intersect with misplaced spots. Is is therefore satisfactory to refine the local minimums and center them in between the peaks as described in the following section.

5.3.3 Refinement of Valley Coordinates

The proposed refinement procedure is to first remove outliers in X^- and Y^- and center all detected local minimums in between the peaks in X^+ and Y^+. In Figure 5.10 is an example when this proposed method is applied on the detected valleys in Figure 5.9.

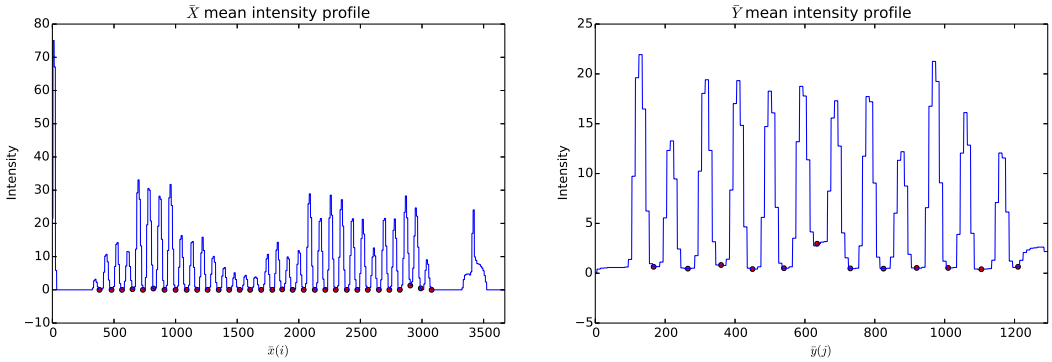


Figure 5.10: The local minimums are plotted as red points. Here, outliers are removed and the minimums are centered in between the peaks. However, two local minimums are not detected (the very first valley in both $\bar{X}$ and $\bar{Y}$.

5.3.3.1 Remove Outliers

Outliers can be detected by analyzing the distances between valley coordinates in X^- and Y^-. If the distance between a point to another is larger than the median distance times a distance threshold T_d, the point is detected as an outlier and is removed. The median distances $\bar{d_x}$ and $\bar{d_y}$ in X^- and Y^- can be calculated as

$$\begin{aligned} \bar{d_x} &= \text{median}(\{x_1^- - x_0^- \,, \ldots \,, x_n^- - x_{n-1}^-\}) \,, \\ \bar{d_y} &= \text{median}(\{y_1^- - y_0^- \,, \ldots \,, y_n^- - y_{n-1}^-\}) \,. \end{aligned} \tag{5.9}$$

5.3.3.2 Center Local Minimums

Each local minimum that is left after removing all outliers, is centered in between its two neighboring peaks, except for the very last and very first valleys that only have one neighboring peak. For the case when a local minimum has two neighboring peaks, the peak indices are calculated as the index of the maximum value one median distance backwards and one median distance forwards of the local minimum. To give an example, for the local minimum x_1^-, the index i_0 of the maximum value in $\bar{X}[x_1^- - d_x : x_1^-]$ and the index i_1 of the maximum value in $\bar{X}[x_1^- : x_1^- + d_x]$ are computed. x_1^- is then replaced with the index that is in between those two indices, which is calculated as

$$i_{\text{new}} = i_0 + \text{round}\left(\frac{i_1 - i_0}{2}\right) . \tag{5.10}$$

For the case when a local minimum only has one neighboring peak, there are two cases: (1) when the neighboring peak has a lower index than local minimum and (2) when the neighboring peak has a higher index than the local minimum. For case (1), the new local minimum is calculated as the the median distance subtracted from the index i_1 of the maximum value one median distance backwards of the local minimum as

$$i_{\text{new}} = i_1 - \bar{d}_x . \tag{5.11}$$

Analogously for case (2), the new local minimum is calculated as the the median distance added to the index i_0 of the maximum value one median distance forwards of the local minimum as

$$i_{\text{new}} = i_0 + \bar{d}_x . \tag{5.12}$$

5.3.4 Add Missing Valleys

The final step is adding missing valley points that were not detected, if any. The proposed method for this is to use the set of edge valleys $\{x_0^- , x_n^-\}$ and $\{y_0^- , y_n^-\}$ and search for peaks in the regions defined by $r_{x0} = [x_0^- - \bar{d}_x , x_0^-]$, $r_{x1} = [x_n^- , x_n^- + \bar{d}_x]$ and $r_{y0} = [y_0^- - \bar{d}_y , y_0^-]$, $r_{y1} = [y_n^- , y_n^- + \bar{d}_y]$. The maximum intensity value is calculated in each region and the region that has the largest maximum is used for calculating the new valley points. The new valley point is calculated as in equation (5.11) if the first region was most significant or as in equation (5.12) if the second region was most significant.

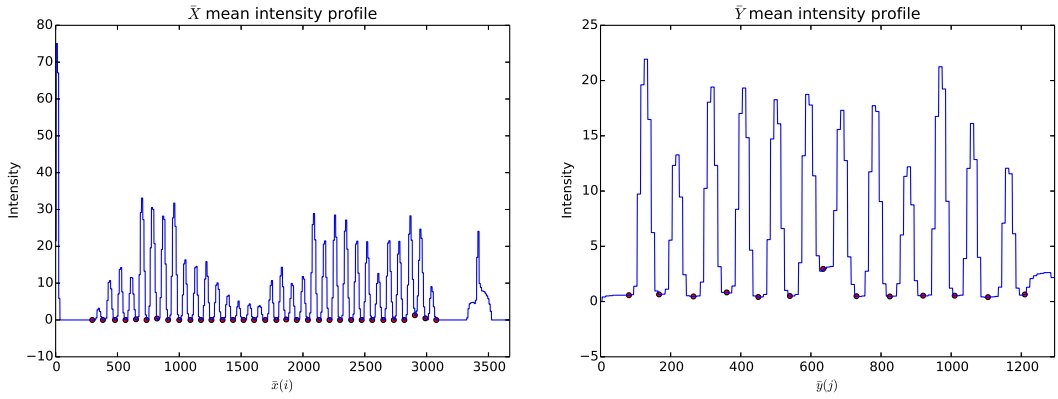


Figure 5.11: The local minimums are plotted as red points. Here, the missing minimums in Figure 5.10 are added.

When all missing local minimums are found, a global grid can be generated from the valley coordinates as the example in Figure 5.12.

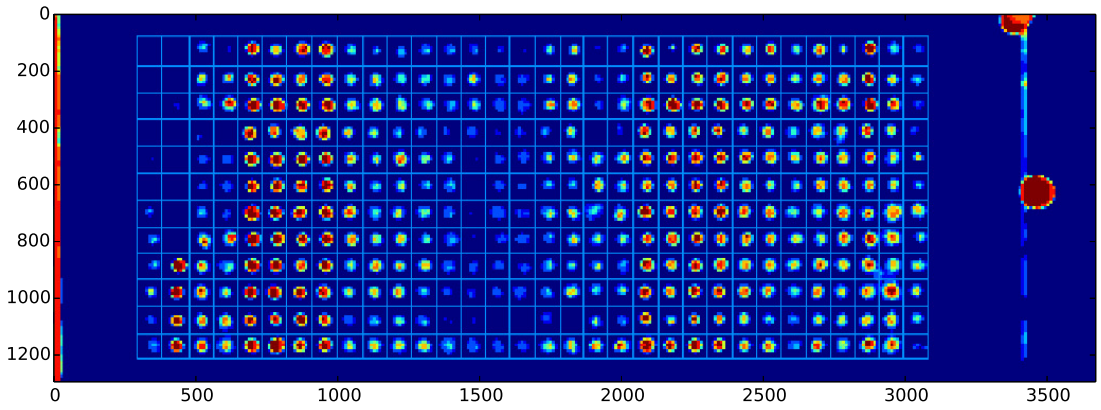
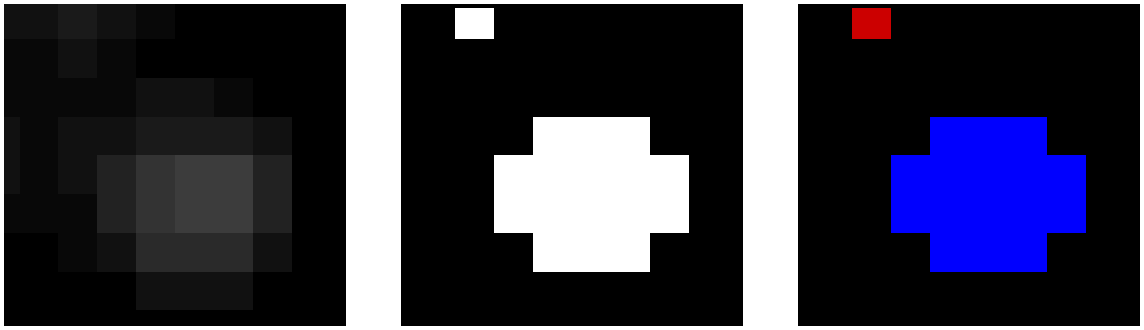


Figure 5.12: An example of a gridded classification map.

5.4 Spot Detection in Grid Regions

The final step in the spot detection framework is to threshold each grid region with Otsu's method and compute the centroid coordinate of the connected component within each thresholded image and use the centroid coordinate as spot coordinate. If the *print type* is double-sided, the previously computed centroids are used to avoid re-computation. In some cases, more than one connected component might be found in a grid region, as the example in Figure 5.13.



(a) An example grid region in a classification map.

(b) The grid region thresholded with Otsu's method.

(c) Two connected components are found in the grid region, marked with red and blue labels.

Figure 5.13: An example with multiple connected components in a grid region.

Multiple connected components can be found in some cases when the spots are dis-

placed, when the alignment of grid regions are not perfectly centered between spots or when contamination and distortions are not successfully removed by the k-NN segmentation. The fundamental problem is to choose the correct component among the set of found components. The proposed method for selection is to compare the height and width of each component and pick the component whose size is the closest to the median size of all global components that are found in a classification map.

5.4.1 Compute Spot Radius

When all computed centroid coordinates are assigned to all spots, a global spot radius is calculated as

$$r = \text{round}\left(\frac{\tilde{w} + \tilde{h}}{2}\right), \tag{5.13}$$

where $\tilde{w}$ is the median component width, $\tilde{h}$ is the median component height and round$(\cdot)$ rounds the element to the nearest integer. An example of a spot detection result from the grid in Figure 5.12 can be seen in Figure 5.14.

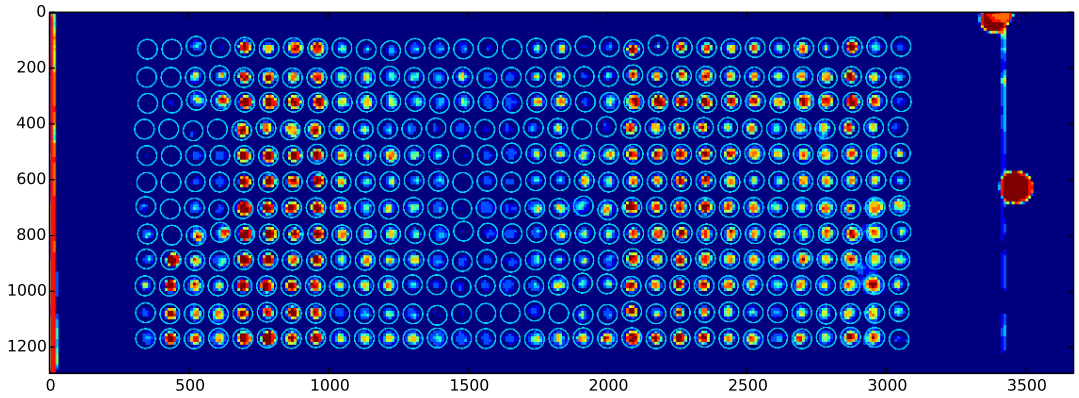


Figure 5.14: Spot detection result of the classification map in Figure 5.12.

5.4.2 Generate Spot Images from Global Array Image

When the centroids are computed as well as the spot radius, each experiment spot can be extracted and saved in a *spot image*. The data in each *spot image* has to be read from the *component images* in the *array image*, which can be achieved by using the methodology in Section 3.2.2. Before a *spot image* can be extracted, the global bounding box coordinates of a spot has to computed, which can be done as

$$
\begin{aligned}
x_1^i &= round((C_x^i - r \cdot \lambda) \cdot M)\,, \\
x_2^i &= round((C_x^i + r \cdot \lambda) \cdot M)\,, \\
y_1^i &= round((C_y^i - r \cdot \lambda) \cdot M)\,, \\
y_2^i &= round((C_y^i + r \cdot \lambda) \cdot M)\,,
\end{aligned}
\tag{5.14}
$$

where M is the decimation rate the *array image* was downsampled with and λ is a *spot extension* parameter. In cases where the centroids are not perfectly centered, a spot extension $\lambda > 1$ can be used to make the spots fit inside the spot images. A drawback with a large λ is larger spot images which is costly, λ should therefore be chosen as close to 1 as possible.

6
Results

In this chapter, the final framework structure is presented as well as a detailed evaluation of the spot segmentation and spot detection performance in various measures.

6.1 Final Framework Structure

The detection program is developed in Python and consists of the data structuring framework, the image processing framework and the spot detection framework described in Chapters 3, 4 and 5. An overview flowchart of the whole framework structure is illustrated in Figure 6.1.

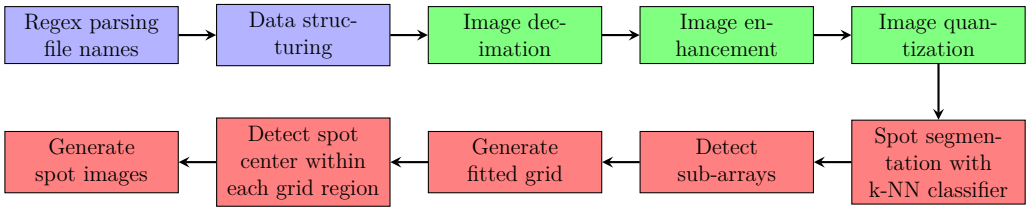


Figure 6.1: Overview flowchart. The black arrows indicates in which order each step is processed. The blue boxes is the data structuring framework, the green is the image processing framework and the red is the spot detection framework.

6.2 Chosen Parameter Values

All parameter values used throughout this thesis are found in Table 6.1.

Parameter	Description	Value
M_{10x}	Decimation rate	25
k	k-NN hyperparameter	10
s	Sampling step size	10 px
z	Sampling zero padding	16 px
T_f	Filter threshold	1.6
T_d	Distance threshold	1.5
Δ	Difference threshold	2
λ	Spot extension	1.4

Table 6.1: Chosen design parameters.

The decimation rate M was chosen as the integer value closest to 10% of the average spot radius in pixels for Persomics' dataset, which was deemed to be sufficient in

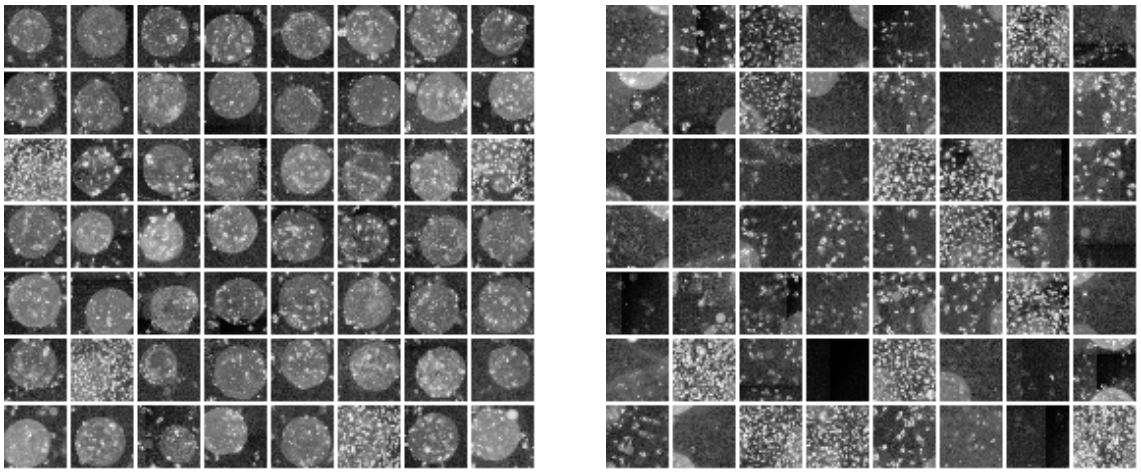
order to represent the locations of spots. For *component images* screened at 10X, M was determined as a value of 25 since the average spot radius was close to 250px. No rigorous test were performed in order to find an optimal k-NN hyperparameter or filter threshold T_f. The sampling step size s was chosen as 10px so the classification maps had at least five sample regions over each spot diameter, which turned out to be sufficient data for gridding. A step size smaller than 10 did not yield any significant improvement on the spot detection result. Moreover, the distance threshold T_d was chosen as a ratio greater than (but close to) the maximum offset from the median distances between rows and columns in *array images*. In cases where the threshold was lower than the maximum offset, true valleys were filtered out in arrays with large rows or columns offsets. Furthermore, the difference threshold Δ was empirically determined as a value of 2, which was the Δ resulting in the highest amount of true valleys. A too low Δ resulted in many falsely detected valleys and a too high one resulted in many missing true valleys. The effects of choosing a few different difference thresholds are presented in Appendix B.1. Finally, the spot extension λ was set to 1.4 which was enough to make poorly detected spots to fit inside the generated spot images.

6.3 Training Database

The tile images in the training database were semi-automatically generated in the following way. A computer program generated random positions and extracted tiles with the size of 32 x 32 px from three different downsampled and enhanced *array images* (from print p67, p68 and p74). Each tile whose region overlapped more than 80% with a spot or background area in the ground truth was included into the training database. The ratio of 80% was empirically determined. Some tiles were manually discarded that contained contamination or over-exposed regions. After this process, 81 background tiles and 75 spot tiles were stored in the training database. The amount of tiles turned out to be sufficient and no rigorous test were performed to find a minimum size of the training database. Some of the tiles that were chosen are visualized in Figure 6.2.

6.4 Spot Segmentation Performance

By viewing actual segmentations from *array images* in Persomics' dataset, a better understanding of the results might be provided. A somewhat good segmentation is found in Figure 6.3a. Here it can be seen that all spots are successfully segmented. There is, however, an edge distortion that is interpreted as a spot which can be seen in the left part of the image. There is also a small contamination in the lower right part that is segmented as a spot. Most of the segmentations of *array images* in Persomics' dataset yields similar performance as this example, but there are a few outliers. An incredibly poor segmentation is found in Figure 6.3b. Here, about a quarter of the spots are not segmented at all and many spots are barely segmented. Furthermore, there are features and contamination in the background that are interpreted as spots. As can be seen in the original image, the image quality of the spots can vary a lot, where some spots are barely visible to the human eye



(a) **(b)**

Figure 6.2: Some examples of images stored in the training database: (a) tiles of spot class and (b) tiles of background class.

and some spots are smeared or noisy.

6.5 Spot Detection Performance

The detection performance of the spot detection is evaluated by applying it to the full Persomics image set of 68 *array images* and comparing the results to a ground truth. The performance is also compared with the current detection framework developed by Persomics, that was presented in related work. An experiment spot is counted as detected if a predicted spot circle covers at least 80% of the spot, if less, it is counted as not detected. Moreover, a predicted spot is counted as correct prediction if 100% of it covers an experiment spot. An example of how predictions are evaluated is illustrated in Figure 6.4.

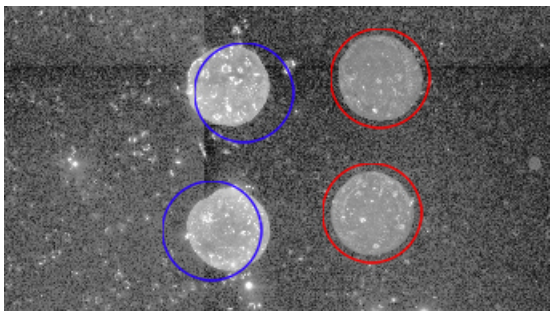
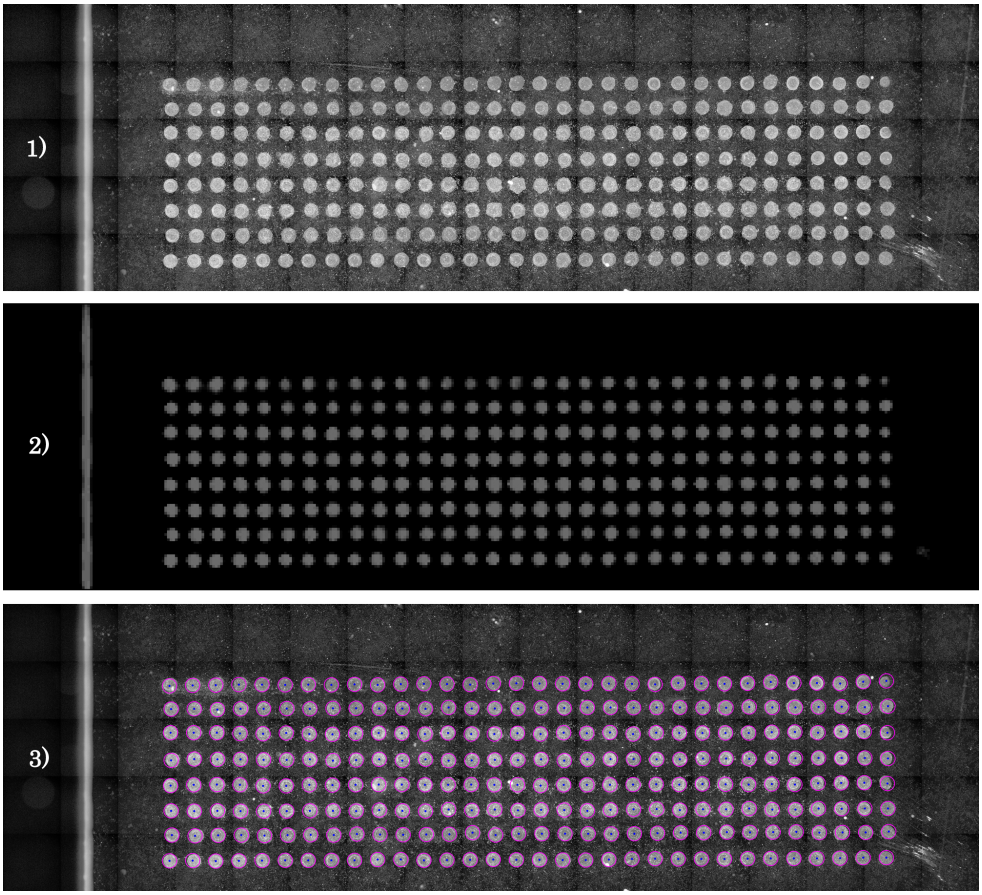
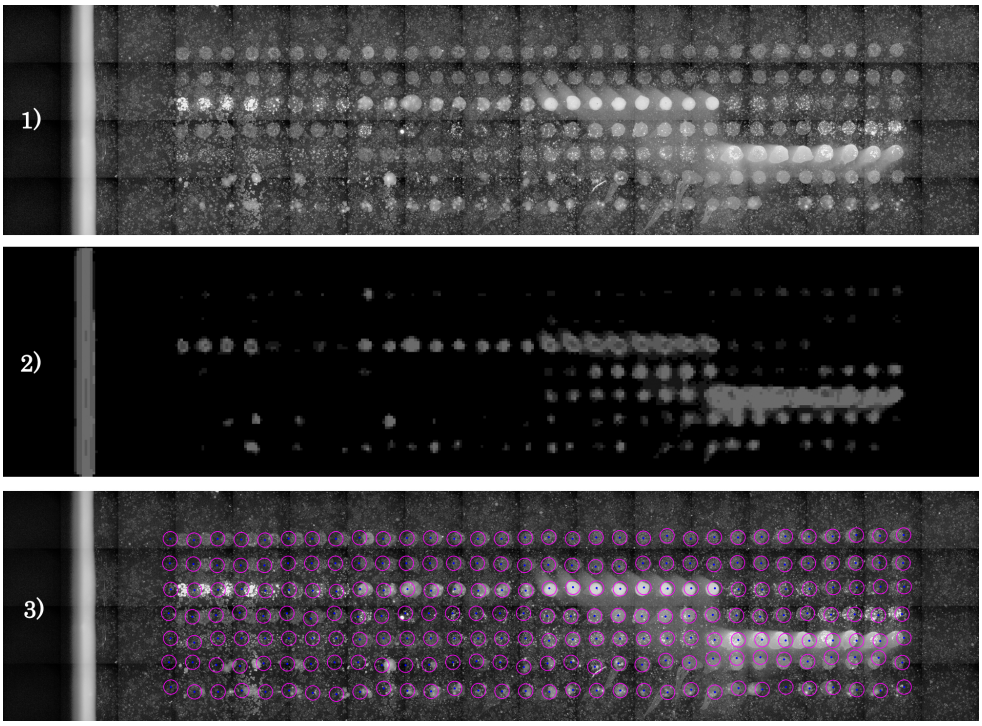


Figure 6.4: Examples of spot detection predictions. In the blue circles to the left, the spots are counted as detected and in the red circles to the right, the spots are counted as both detected and correctly predicted.

Based on this, the precision of the spot detection is computed and the results are presented in Table 6.2 and Figure 6.5. An example of a good and a poor spot detection is provided in Figure 6.3a and 6.3b respectively.



(a) An example array (1) with good segmentation (2) and detection (3).



(b) An example array (1) with poor segmentation (2) and detection (3).

Figure 6.3: Segmentation and detection results for a p67 (a) and a p83 array (b).

Print number	Total Spots	Detected spots		Correct prediction	
		Our method	Old method	Our method	Old method
p67	512	512	512	507	448
p68	1280	1280	0	1247	0
p69	512	512	0	501	0
p70	2560	2560	1605	2554	1585
p71	448	448	448	448	375
p74	3072	3072	2304	2545	2263
p76	320	320	320	320	257
p79	64	63	44	36	16
p80	800	641	27	637	4
p81	320	320	62	316	7
p82	768	768	0	766	0
p82-d	3072	3072	0	2999	0
p83	224	191	181	73	18
p84	3040	3040	1938	3008	1684
p84-d	6080	6080	3912	6057	2561
p85	3648	3648	1368	3633	1023
Sum	26720	26527 (99.27%)	12721 (47.61%)	25647 (95.98%)	10241 (38.33%)

Table 6.2: Summary of the detection results. "Our method" is the framework presented in this thesis and "Old method" is the framework developed by Persomics.

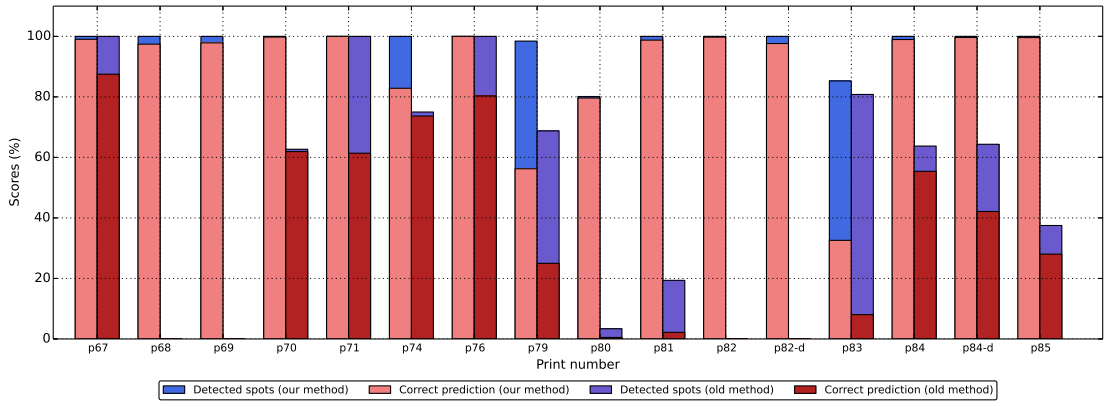


Figure 6.5: Summary of the detection results. "Our method" is the framework presented in this thesis and "Old method" is the framework developed by Persomics.

Framework	Time [ms] per spot	
	Our method	Old method
Data structuring	0.021	–
Image decimation	22.53	–
Image enhancement	1.18	2.44
Image quantization	0.094	0.094
Spot segmentation	28.90	1.20
Sub-array detection	2.08	–
Grid fitting	8.83	–
Spot detection	0.54	121.35
Generate image files	56.67	908.73

Table 6.3: Average run-time per spot for different frameworks. "Our method" is our framework and "Old method" is the framework developed by Persomics.

6.6 Computational Performance

The computational performance of each framework is evaluated by computing the average run-time on the *array images* per spot. The computational run-time results are presented in Table 6.3 for both our framework and the framework developed by Persomics.

7

Discussion and Conclusions

The following chapter describes and presents future improvements and conclusions about the study.

7.1 Discussion and Future Improvements

Even though the methodology provided in this study is fully functional for many *array images*, there are still ways to improve the spot segmentation and spot detection. One improvement would be to add more tiles from different *array images* to the training database to improve the segmentation results. That would, however, result in larger computational complexity of the segmentation. Due to this, it is worth considering using another classifier that might have a better computational performance, such as deep neural networks that was brought up in related works. Neural networks are very expensive to train, but once trained, it is relatively cheap to classify tiles, which clearly is an advantage. To improve the spot detection, one way would be to tune the design parameters even further in order to find optimal values.

Moreover, as can be seen from the results, two of the most computationally intensive parts of the detection program is the spot segmentation and image decimation. As one easily can see, both approaches processes images that are independent of the others, which means that the processing could be done in parallel to improve the computational performance.

Another remark is that the developed framework cannot handle disoriented array images that are rotated, which is one of the problems specified in the research questions. A disadvantage with single-sided *print type* is that orientation cannot be detected and compensated for, which is why a solution to this problem was not covered in this study. To detect orientation, *orientation markers* have to be added to the *print map*, which are distinguishable spot patterns (for instance intentionally removed spots and varying spacings between spots). This have been discussed with the company and will be added to upcoming *array plate* prints. An advantage with the double-sided *print type* is that it does not need any orientation markers which is due to the rotation symmetry of spots in the *print map*.

Finally, it is worth evaluating the sub-array detection thoroughly on more image data since it was only tested on nine double-sided arrays, which is insufficient in order to draw any accurate conclusions.

7.2 Conclusions

In this study, a program for spot detection in fluorescence microscopy images has been designed, implemented and tested. It has been shown that the spot detection program can handle *component images* of different magnification as input, as well as outperforming the currently used method, which relies on pre-saved annotation coordinate data for spots. The framework is capable of detecting sub-arrays, but it cannot handle disoriented array images that are rotated. Furthermore, it manages to segment spots quite well, but at the price of a relatively high computational complexity since classifying a tile requires a comparison to every single tile in the training database. The spot detection fails for some cases as a result of a poor segmentation.

Bibliography

[1] Persomics, Home page, http://www.persomics.com, accessed: 2017-03-14.

[2] Persomics, Build research tools for cutting-edge biotech, http://www.chalmers.se/en/departments/s2/education/masters-programmes/Documents/Build%20research%20tools%20for%20cutting-edge%20biotech.pdf, accessed: 2017-01-28.

[3] Persomics, Persomics Lab Protocol Template, http://www.persomics.com/hubfs/Documents/Persomics_Protocol_Template.pdf, accessed: 2017-05-09.

[4] I. Smal, M. Loog, W. Niessen, E. Meijering, "Quantitative comparison of spot detection methods in fluorescence microscopy", IEEE Trans. Med. Imag., vol. 29, no. 2, pp. 282–301 (Feb. 2010).

[5] K. Prasanna, W. Dick, A. Thomas, "Threshold selection using minimal histogram entropy difference", Opt. Eng., vol. 36, no. 7,pp. 1976–1981 (Jul. 1997).

[6] N. Otsu, "A threshold selection method from gray-level histograms", IEEE Trans. Syst., Man, Cybern., vol. 9, no. 1, pp. 62–66 (Jan. 1979).

[7] R. Stoklasa, P. Matula, "Road Detection Using Similarity Search", In Proceedings of 2nd International Conference on Robotics in Education (RiE 2011). Vienna, Austria, September, pp. 95-102. INNOC - Austrian Society for Innovative Computer Sciences. (2011).

[8] A. Gavlasová, A. Procházka, M. Mudrová, "Wavelet Based Image Segmentation", Proceedings. of 14th Annual Conference Technical Computing 2006, pages GPM/1-7 (Jan. 2006).

[9] E. Shelhamer, J. Long, T. Darrell, "Fully Convolutional Networks for Semantic Segmentation", CoRR, abs/1411.4038 (May. 2016).

[10] A. Karpathy, Image Classification: Data-driven Approach, k-Nearest Neighbor, train/val/test splits, http://cs231n.github.io/classification/, accessed: 2017-05-08.

[11] L. Rueda, V. Vidyadharan, "A Hill-Climbing Approach for Automatic Gridding of cDNA Microarray Images", IEEE Transactions on Computational Biology and Bioinformatics, 72–83. 10.1109/TCBB (Mar. 2006).

[12] J. Angulo, J. Serran, "Automatic Analysis of DNA Microarray Images Using Mathematical Morphology", Bioinformatics 19(5):553–562. 10.1093/bioinformatics/btg057 (Apr. 2003).

[13] L. Rueda, I. Rezaeian, "A fully automatic gridding method for cDNA microarray images", BMC Bioinforma. 12:1–17 (2011).

[14] G. Rafael, W. Richard, Digital Image Processing, 3rd Edition, Prentice-Hall, 2008.

[15] xytrax, Regular Expressions - User Guide, http://www.zytrax.com/tech/web/regex.htm#brackets, accessed: 2017-05-10.

[16] Python, 6.2. re - Regular expression operations, https://docs.python.org/3/library/re.html, accessed: 2017-05-10.

[17] Python, 5. Data Structures, https://docs.python.org/2/tutorial/datastructures.html, accessed: 2017-05-18.

[18] Python, 9. Classess, https://docs.python.org/2/tutorial/classes.html, accessed: 2017-05-18.

[19] Python, 8.7. sets — Unordered collections of unique elements, https://docs.python.org/2/library/sets.html, accessed: 2017-05-18.

[20] Wikipedia, Floor and ceiling functions, https://en.wikipedia.org/wiki/Floor_and_ceiling_functions, accessed: 2017-05-18.

[21] Wikipedia, Decimation (signal processing), https://en.wikipedia.org/wiki/Decimation_(signal_processing), accessed: 2017-05-18.

[22] J. A. Parker, R. V. Kenyon, D. E. Troxel, "Comparision of Interpolating Methods for Image Resampling", IEEE Transactions on Medical Imaging, vol. MI-2, NO. 1 (Mar. 1983).

[23] Wikipedia, Image scaling, https://en.wikipedia.org/wiki/Image_scaling, accessed: 2017-05-16.

[24] A. Tanbakuchi, Comparison of OpenCV Interpolation Algorithms, http://tanbakuchi.com/posts/comparison-of-openv-interpolation-algorithms/, accessed: 2017-05-13.

[25] ImageJ, ContrastEnhancer plugin, https://imagej.nih.gov/ij/download/tools/source/ij/plugin/ContrastEnhancer.java, accessed: 2017-02-19.

[26] S. R. Kulkarni, Lecture Notes for ELE201 Introduction to Electrical Signals and Systems: Sampling and Quantization (1999), https://sisu.ut.ee/sites/default/files/imageprocessing/files/digitizn.pdf, accessed: 2017-05-06.

[27] T. Kong, A. Rosenfeld, Topological Algorithms for Digital Image Processing, North Holland, 1996.

[28] A. Shack, Connected Component Labelling), http://aishack.in/tutorials/connected-component-labelling/, accessed: 2017-05-18.

[29] OpenCV, Structural Analysis and Shape Descriptors, http://docs.opencv.org/3.1.0/d3/dc0/group__imgproc__shape.html, accessed: 2017-05-16.

[30] Wikipedia, Image moment, https://en.wikipedia.org/wiki/Image_moment, accessed: 2017-05-02.

[31] OpenCV, Structural Analysis and Shape Descriptors, http://docs.opencv.org/3.0-beta/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html, accessed: 2017-05-02.

[32] L. Miao, "Comparative Analysis of Two Clustering Algorithms: K-means and FSDP (Fast Search and Find of Density Peaks)", SJSU ScholarWorks, Master's Projects. Paper 427 (2015).

[33] scikit learn, K-Means clustering, http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html, accessed: 2017-05-16.
[34] MATLAB, Intensity Profile of Images, https://se.mathworks.com/help/images/intensity-profile-of-images.html, accessed: 2017-05-02.
[35] E. Billauer, peakdet: Peak detection using MATLAB, http://billauer.co.il/peakdet.html, accessed: 2017-03-18.

A

Implementation in Python

A.1 Data Structuring

Following Python code parses component image file names from Persomics and stores the metadata in a Component class.

```
class Component:
    def __init__(self, filename, name, file_type, col, row, channel):
        self.file = filename
        self.name = name
        self.file_type = file_type
        self.col = col
        self.row = row
        self.channel = channel

regex = '(?P<Row>[A-Za-z]{1,*})(?P<Col>[0-9]{1,2})_0' +
    '_(?P<Channel>[0-9])_RFP_001\.(?P<Type>[A-Z]{3})'

cols, rows, channels, components = [], [], [], []
for file in files:
    match = re.search(regex, file)
    cols.append(match.group('Col'))
    rows.append(match.group('Row'))
    channels.append(int(match.group('Channel')))

name = match.group('Name')
file_type = match.group('Type')

col_unique = sorted(set(cols))
row_unique = sorted(set(rows))
channel_unique = sorted(set(channels))

col_indices = [col_unique.index(c) for c in cols]
row_indices = [row_unique.index(r) for r in rows]
channel_indices = [channel_unique.index(ch) for ch in channels]

for i in range(0, len(files)):
    components.append(Component(filename, name, file_type, col, row, channel))
```

A.2 Read from Component Images

To read data from a set of component images in Python, one way is to create an empty 2D-array image with height and width of the region of interest calculated as

$$h_{\text{roi}} = X_e - X_s + 1 \quad w_{\text{roi}} = Y_e - Y_s + 1 \tag{A.1}$$

where 1 is added to the height and width due to indexing. The next step is then to loop though all the indices in R and C, read the component images that have those indices, and with the help of Equation 3.4 and 3.5 extract the data and store it into the empty 2D-array image. Following Python code reads the region of interest from any set of component images, where the input component_paths is a 2D-array of file paths to the component images:

```
import numpy as np
import tifffile
from math import floor
def read_component_data(X_s, X_e, Y_s, Y_e, w, h, component_paths):
    r_s, r_e = int(floor(X_s / h)), int(floor(X_s / h))
    c_s, c_e = int(floor(Y_s / w)), int(floor(Y_e / w))
    h_roi, w_roi = X_e - X_s + 1, Y_e - Y_s + 1
    img = np.zeros((w_roi, h_roi))
    x_s_roi = 0
    for row in range(r_s, r_e + 1):
        y_s_roi = 0
        for col in range(c_s, c_e + 1):
            if Y_s > col * w: y_s = Y_s - col * w
            else: y_s = 0
            if X_s > row * h: x_s = X_s - row * h
            else: x_s = 0
            if Y_e > (col + 1) * w: y_e = w
            else: y_e = Y_e - (col) * w
            if X_e > (row + 1) * h: x_e = h
            else: x_e = X_e - (row) * h
            im = tifffile.imread(component_paths[row][col])
            im = im[x_s:x_e, y_s:y_e]
            img[x_s_roi:x_s_roi + (x_e - x_s),
                y_s_roi:y_s_roi + (y_e - y_s)] = im
            y_s_roi += (y_e - y_s)
        x_s_roi += (x_e - x_s)
    return img
```

B

Results

B.1 Effects of different difference thresholds

In Figure B.1 and B.2, the found valleys in two intensity profiles are visualized for different difference thresholds Δ.

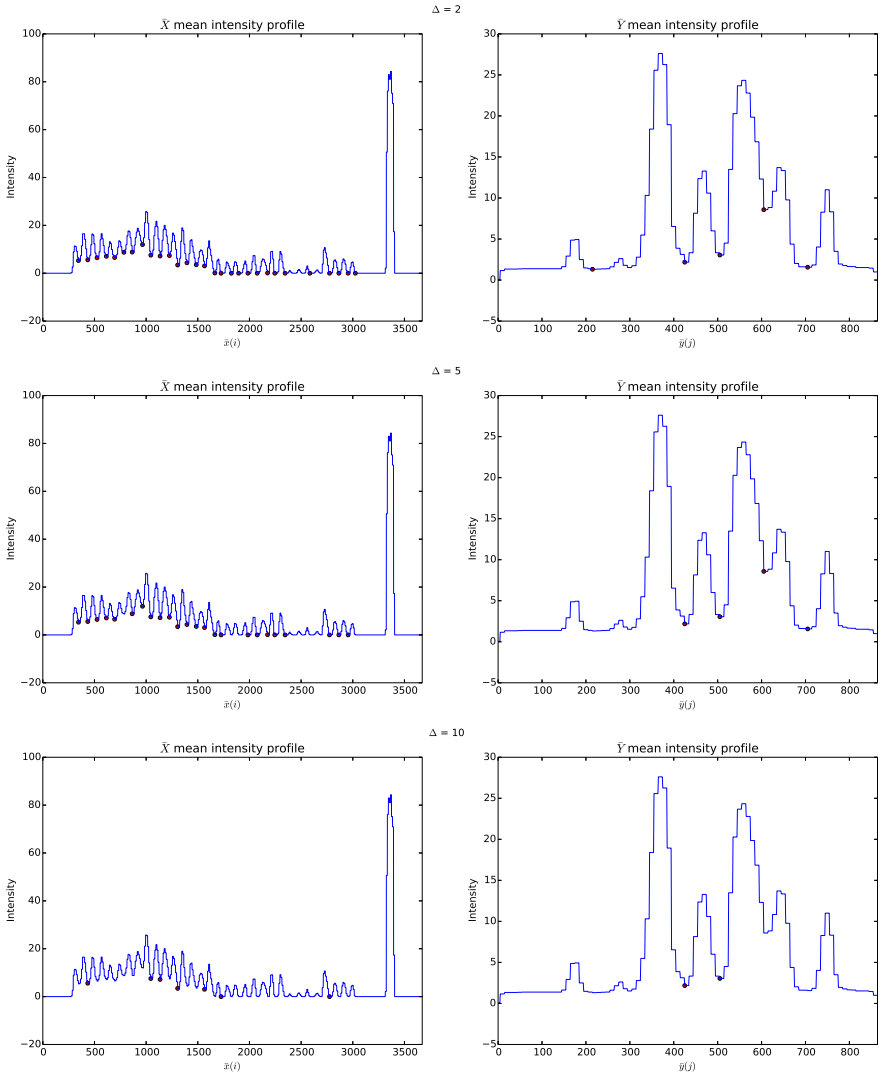


Figure B.1: Found valleys for different difference thresholds Δ.

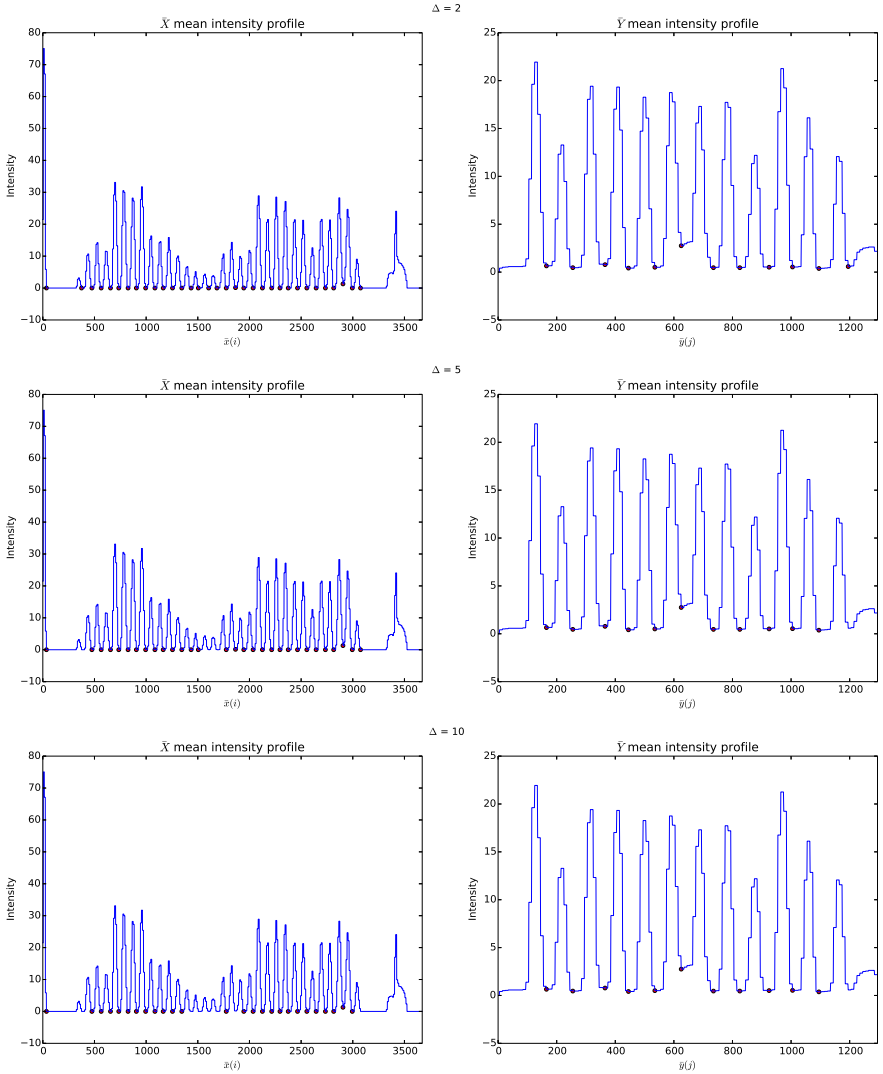


Figure B.2: Found valleys for different difference thresholds Δ.